OKTATÁSI HIVATAL

NAT 2020

11

Digitális kultúra

tankönyv

DIGITÁLIS KULTÚRA 11.

OKTATÁSI HIVATAL

A kiadvány 2022. 02. 22-től 2027. 08. 31-ig tankönyvi engedélyt kapott a TKV/72-7/2022. számú határozattal.
A tankönyv megfelel a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról szóló 110/2012. (VI. 4.) Korm. rendelet alapján készült, 2020. 01. 31. után kiadott, 9–11. évfolyam digitális kultúra tantárgy kerettantervének.

A tankönyvvé nyilvánítási eljárásban közreműködő szakértő: Györgyi Tamás

Tananyagfejlesztők: Abonyi-Tóth Andor, Farkas Csaba, Fodor Zsolt, Jeneiné Horváth Kinga, Reményi Zoltán, Siegler Gábor, Varga Péter

Kerettantervi szakértő: Siegler Gábor
Lektor: Farkasfalvy Judit
Szerkesztő: Széll Szilvia
Fotók: Shutterstock (31 posztos)

ISBN 978-963-436-291-3

Oktatási Hivatal
1055 Budapest, Szalay utca 10–14.
Telefon: (+36-1) 374-2100
E-mail: tankonyv@oh.gov.hu

Ez a tankönyv a Széchenyi 2020 Emberi Erőforrás Fejlesztési Operatív Program EFOP-3.2.2-VEKOP-15-2016-00001 számú, „A köznevelés tartalmi szabályozóinak megfelelő tankönyvek, taneszközök fejlesztése és digitális tartalomfejlesztés” című projektje keretében készült. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE

Tartalom

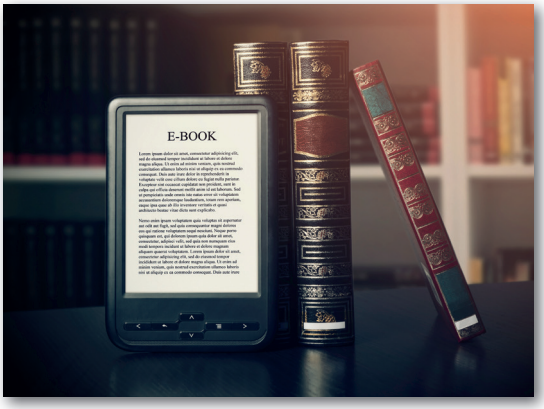
Előszó

Kedves Diákok!

Környezetünkben ma már digitálisan történik a legtöbb esemény, kommunikáció, ügyintézés, kapcsolattartás a családunkkal, az iskolával, a közintézményekkel, a közszolgáltatókkal és a hétköznapi élet számos egyéb szereplőjével. A mesterséges intelligencia alapja, hogy bár a nagy mennyiségű adatgyűjtés az okosrendszerek kiindulópontja, de ahhoz, hogy a kérdéseinkre időben választ kapjunk, nélkülözhetetlen az adatok valós idejű, automatikus feldolgozása. Környezetünk tele van szenzorokkal, amelyek az okoseszközök bemenetei, és hihetetlen mennyiségű adatot gyűjtenek, küldenek és tárolnak.

Nem lesz mindenkiből – sőt feltehetően csak kevesekből – az információ feldolgozásával foglalkozó szakember, de a digitális világ működésének alapjait mindannyiunknak ismernünk és értenünk kell. Ahhoz, hogy értsük digitális környezetünket, tudnunk kell, hogy mit várhatunk el a digitális megoldásoktól, mennyire megbízhatóak, és milyen automatizmusokat tartalmaznak.

Az idei tanév témáinak középpontjában változatlanul az adat, az adatfeldolgozás és az adattovábbítás áll.



A tankönyv első részében újra a dokumentumkészítéssel foglalkozunk, de most már külön hangsúlyt kap a nagy mennyiségű szöveg és adat feldolgozása. A léptékváltás a feldolgozási módszer változását jelenti. Lényeges, hogy a dokumentumkészítés immár több szereplő részvételével történik: több szerző – vagy szerző és lektor – dolgozik együtt, így fontossá válik a változtatások, a verziók követése.

A második részben folytatjuk a táblázatkezelés módszereinek megismerését, bővítjük ismereteinket, nagyobb adatmennyiségekkel dolgozunk. Különösen fontos lesz a nagy mennyiségű adatokban való keresés, szűrés és a függvények célszerű használata.

A harmadik részben az adatbázis-kezelés eszközeivel, módszereivel, elméletének alapjaival és az adatkezelés gyakorlatával ismerkedünk meg. Az adatbázis rendszerezetten tárolt adatok összessége, amelyen kérdéseink alapján olyan műveleteket végezhetünk, melyek választ szolgáltatnak.

A következő, rövidebb fejezetekben az online kommunikáció módszereit, az információs társadalom különböző kérdéseit és ehhez kapcsolódóan a mobilhálózati alkalmazások eszközeit, használatát tárgyaljuk. Az előzetes, remélhetőleg gyarapodó ismereteinket rendszerezzük, bővítjük.

Ezt követően az algoritmusok és programkészítés témakörében folytatjuk tanulmányainkat, és bővítjük ismereteinket. A programozás alapfogalmait, módszereit több feladat, illetve feladatsor megoldásához felhasználjuk majd. A feladatok megoldásához a tizedikes tankönyvben már alapjaiban megismert Pythont használjuk, de fontos ismét hangsúlyoznunk, hogy a választott nyelv csupán egy eszköz, amelyet most (vagy később) más elterjedt nyelv(ek) válthat(nak) fel.

Könyvünk utolsó fejezetében, hasonlóan az előző kötetekhez, az informatikai eszközök használatáról olvashatunk. Ezt a részt ne önállóan dolgozzuk fel, hanem tartalmi elemeit kisebb részletekben, a többi témakör tárgyalásakor megjelenő fogalmakhoz kapcsolva lapozzuk fel!

A tankönyv szerves részét képezik a tananyaghoz kapcsolódó elektronikus anyagok, fájlok, amelyek a https://tankonyvkatalogus.hu/site/kiadvany/OH-DIG11TA oldalról tölthetők le. A fejezetenként külön mappákba rendezett forrásfájlokat tömörítve tudjuk letölteni. Csomagoljuk ki, és mentsük el egy külön mappába a gépünkön, így az adott feladatnál könnyen megtaláljuk őket. Elkészült munkáinkat szintén gyűjthetjük majd ebben a mappában.

Jó munkát és a tankönyv eredményes használatát kívánjuk!
A szerzők

Szövegszerkesztés

A kilencedik évfolyamon áttekintettük a szövegszerkesztés legfontosabb módszereit, az ezekhez tartozó eszközöket. Tanulmányainkat most a tizenegyedik évfolyamon folytatjuk – az eddigi fogalmak, eljárások, technikák átismétlése után – a nagy méretű, valamint az online, csoportosan szerkeszthető dokumentumokkal.

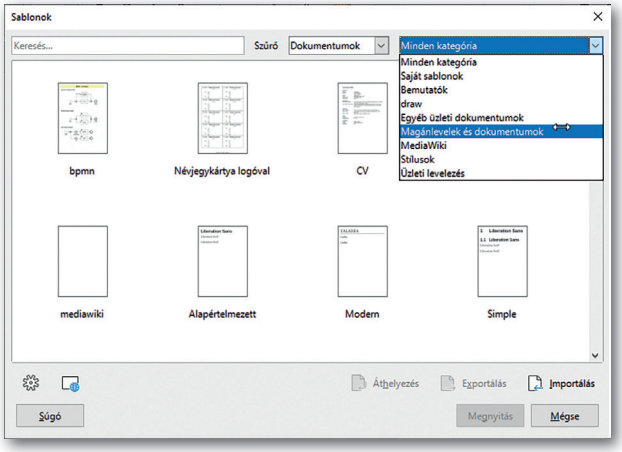
Nagy dokumentumok hatékony szerkesztése

Iskolai munkája során már mindenkinek kellett készítenie 2-3 oldalasnál hosszabb, valamilyen adott szerkezeti meghatározásoknak megfelelő dokumentumot. A hangsúly a szerkezeten van, és nem a karakterek számán. A hatékony szövegformázás és az egységes szerkezet kialakítása a formázások csoportosított használatával, a **stílusok** alkalmazásával érhető el.

A kész stílusok használata ugyan gyors lehet, de az egyéni igények, elképzelések megvalósítására nem alkalmasak. Érdemes vagy a rendelkezésre álló stílusokat módosítani, vagy újakat készíteni. Az igazán sok szöveget, dokumentumot készítők a saját stílusaikat még magasabb szintű formázási struktúrába, **sablonba** rendezhetik, és elmenthetik. Ez lehetővé teszi a stílusok újrafelhasználását és például az egységes céges dizájn kialakítását.

Sablonok

A szövegszerkesztő programok dokumentumtípusonként, illetve használati területenként több, szakemberek által tervezett **sablont** is tartalmaznak. Ezeket az új dokumentum létrehozásakor, indításkor lehet kiválasztani.



► Sablonok témaköreinek gyűjteménye (LibreOffice Writer)

A legtöbb sablon, akárcsak egy kitöltendő űrlap, a formázott szakaszok mellett a tartalmi kitöltést is segíti.

► Önéletrajzsablon tematikus szakaszokkal (Microsoft Word)

Egy új dokumentum mindig valamilyen sablonra épül, s ha nem adunk meg mást, akkor a szövegszerkesztő a *Normál* sablont alkalmazza. Az ebben tárolt és rendelkezésünkre álló stílusok hierarchikus rendszert alkotnak. A *Normál* sablonban történő stílusmódosítás következménye, hogy minden olyan dokumentumra kihat, amelynél nem adunk meg más sablont.

Sablonokat magunk is létrehozhatunk, amelyeket speciális, de visszatérő feladatok megoldásához használhatunk, például állandó elemeket tartalmazó levélpapír, szóróanyag készítéséhez. A dokumentumba a nem változó szöveges és grafikus elemeket elhelyezzük, majd megadjuk a kívánt oldalbeállítást, -méretet, -tájolást, stílusokat és egyéb formázásokat. Mentéskor a dokumentum típusának a *Sablont* állítjuk be. A sablon használata része lehet az iskola, az intézmény vagy egy cég arculati tervének.

Stílusok módosítása és létrehozása

A **stílus** alap- (örökölt) és általunk beállított formátumok gyűjteményét zárja egységbe, azokat nevesíti. Használata akkor előnyös, ha hosszabb, tagolt dokumentumokat készítünk, vagy utólagosan módosítjuk, frissítjük a formátumokat, beállításokat.

A *Normál* sablonban definiált címsorstílusok használatával hozhatunk létre tartalomjegyzéket. A tartalomjegyzéket azonban más stílusok alapján is elkészíthetjük, csak következetesen kell használnunk a formázásokat.

A címsorstílusok hierarchiájában a legfelső szint a *Címsor 1* stílusú cím, ezt követi a *Címsor 2* stílusú alcím, és így tovább az emelkedő sorszámokkal.

A nem megfelelő tulajdonságú formátumstílus beállításait kétféle módszerrel módosíthatjuk:

- Szerkesztéssel: például a *Kezdőlap* > *Stílusok* csoportban (illetve *Stílusok* menüben) a módosítani kívánt stílusnál a *Módosítás* parancsot választjuk. A párbeszédablakban az összes stílusjellemzőt módosíthatjuk.
- Stílusjegyek alapján: az adott stílussal meghatározott szöveget újraformázzuk az elvárásoknak megfelelően. Például a *Stílusok* > *Stílusok alkalmazása* menüpontban az *Ugyanaz* gombot (illetve a *Stílusok* > *Kiválasztott stílus frissítése* menüpontot) választjuk.

1. példa: Tartalomjegyzék készítése és a stílusok módosítása

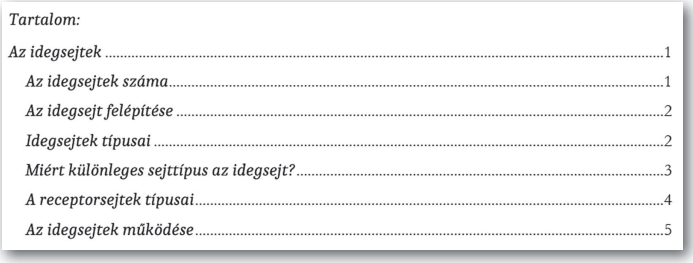
Nyissuk meg a tankönyv weboldaláról letöltött `ideg_alap.docx` nevű fájlt! (A dokumentum szövege és képei a 9–10.-es biológia-tankönyv II. kötetének [OH-BIO910TA/II] 55–58. oldaláról származnak.)

A dokumentum előkészített, részben formázott szöveget tartalmaz, és a képek, ábrák a megfelelő bekezdések előtt találhatók. Több lépésben készítjük el a tanulmány végleges megjelenítését a minták és a leírás szerint.

A forrás, a tartalomnak megfelelő tagolással, címsorokkal – *Címsor 1*, *Címsor 2* és *Címsor 3* – formázva adja a dokumentum szerkezetét.

Feladatok

1. Munkánkat mentsük `ideg_1pelda_mego` néven a szövegszerkesztő alapértelmezett formátumában!
2. Hozzunk létre a dokumentum végén tartalomjegyzéket *„Tartalom:”* címmel a *Címsor 1* és *Címsor 2* bekezdések alapján! A *Címsor 3* formázású alcímek ne szerepeljenek a tartalomjegyzékben!



▶ Tartalomjegyzék beállítása a *Címsor 1* és *Címsor 2* stílusú bekezdések alapján (Microsoft Word)

3. Módosítsuk a *Normál* stílus betűtípusát Caladea vagy más talpas típusra! A betűméret 12 pontos, a sorköz szimpla, az igazítás sorkizárt legyen!
4. Alkalmazzunk a dokumentumban automatikus elválasztást!
5. Módosítsuk a dokumentumban a *Címsor 1*, *Címsor 2* és a *Címsor 3* stílust úgy, hogy a betűtípus a *Normál* stíluséval egyezzen meg! A betűméretek rendre 26, 16 és 14 pontosak legyenek! Kapcsoljuk be az automatikus frissítést, végül állítsuk be, hogy a betűszín automatikus legyen!

6. A dokumentum élőfejében a második oldaltól páros oldalon balra, páratlan oldalon jobbra zártan „Az idegsejtek” felirat jelenjen meg, a szövegtükör szélességében vékony fekete vonallal aláhúzva! Az első oldal fejléce maradjon üresen!
7. A képeket már tartalmazza a dokumentum, de a méretet, az igazítást, a szöveg körbefuttatását és esetleg a képaláírásokat nekünk kell beállítanunk. A képek és a feliratuk elhelyezéséhez használhatunk szegély nélküli táblázatot, vagy alkalmazhatjuk a helyi menü *Felirat beszúrása* menüpontját. A képaláírások szövege kapcsos zárójelben a képek után található. A kapcsos zárójeleket töröljük!

A kép leírása	**Szélesség (cm)**	**Igazítás**	**Körbefuttatás**	**Képaláírás**
Idegsejtek	4	balra	igen	nincs
Elefánt	6	balra	nincs	nincs
Az idegsejtek felépítése	6	jobbra	igen	van
Különböző felépítésű idegsejtek	7	egymás mellett a két kép középen		van
Különböző funkciójú idegsejtek	9			van
Ingerület továbbítása	6	jobbra	igen	van
A jelfeldolgozás sejten belüli folyamata	5	középre	nincs	van
Zenész	5,5	középre	nincs	nincs
A receptorsejtek típusai	8	középre	nincs	van
A nyugalmi potenciál mérése	5,2	egymás mellett a két kép középen		van
A nyugalmi potenciált kialakító Na^+/K^+-pumpafehérje	5,2			van
A sejthártyában lévő ioncsatornák	8,5	középre	nincs	nincs

Korrektúra és véleményezés

Változtatások követése

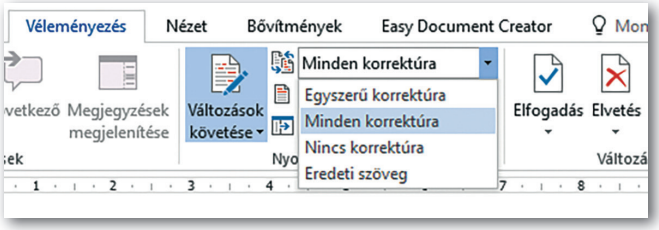
A **korrektúra** a tartalmában elkészített dokumentum ellenőrzése tartalmi, nyelvtani, tördelési és más speciális szempontok alapján.

A „több szem többet lát” közmondás szerint a szerzőt a tartalmában és megjelenésében igényes munka elkészítésében mások megjegyzései, kiegészítései, javításai segíthetik. Egy vagy több, a témához értő szerkesztő, lektor, korrektor a szerzőétől eltérő szemmel vizsgálhatja az írás tartalmát és formátumát. Például azt, hogy:

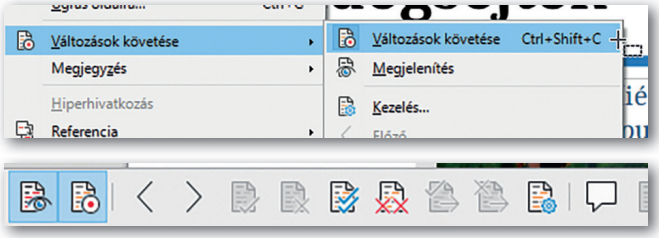
- áttekinthető-e a dokumentum megfelelő címekkel, alcímekkel tagolt szerkezete;
- helyes-e az írásjelek és szimbólumok elhelyezése;
- stilisztikai és helyesírási szempontból megfelelő-e, könnyedén olvasható-e a szöveg;
- egységes-e a kiemelési módszerek, a dőlt és a félkövér betűstílus alkalmazása;
- megfelelő-e a képletek és ábrák helye és helyzete.

A korrektúra funkció használatát igényli az is, ha egy dokumentumon többen fognak dolgozni, és a változtatásokat követhetővé szeretnék tenni.

A korrektúra eszköztára a különböző szövegszerkesztő programokban más-más helyen érhető el, de funkcionalitásuk hasonló.



▶ A *Véleményezés* szalag funkciói (Microsoft Word)

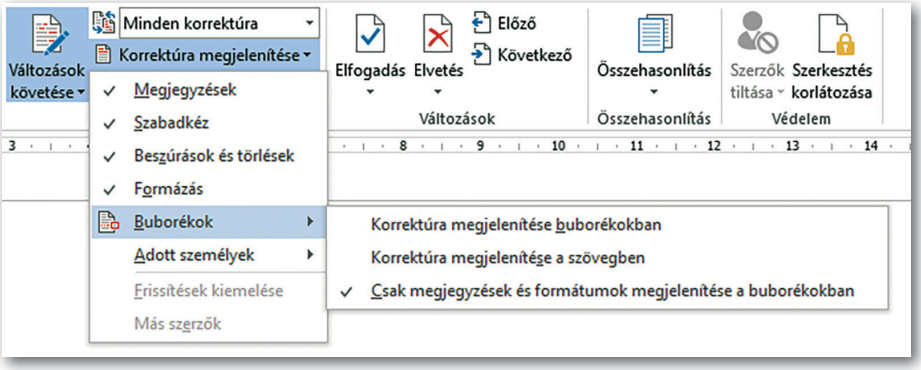


▶ A *Szerkesztés* > *Változások követése* menü és eszköztár (LibreOffice Writer)

A korrektúrához például a *Változások követése* váltógombbal bekapcsoljuk a nyomon követést, majd elvégezzük a szükséges módosításokat a dokumentumon. A korrektúra bekapcsolt állapotában a dokumentum eredeti szövege törlés esetén nem tűnik el, csak áthúzva látszódik, a beszúrt szöveg pedig színes betűvel jelenik meg.

A *Korrektúra* eszköztáron többféle megjelenítés közül választhatunk attól függően, hogy az eredeti, a korrektúrával ellátott vagy a változtatásokat a margón buborékban megjelenítő változatot szeretnénk-e látni. A korrektúra megjelenítésének beállításával a csoportmunkát finomhangolhatjuk.

A korrektúrában tett módosítási javaslatokat a szerző és/vagy szerkesztő a dokumentum frissítése során elfogadhatja vagy elvetheti.

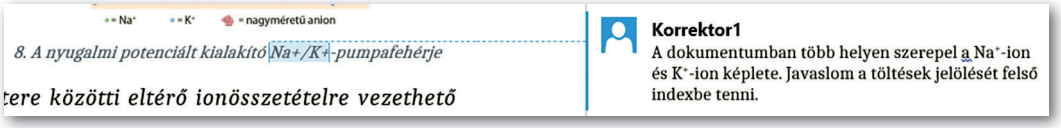


► A korrektúra megjelenítésének beállítása (Microsoft Word)

Megjegyzések használata

A szerkesztett szöveg adott pontjához vagy részletéhez **megjegyzéseket** is fűzhetünk. A megjegyzésben kérdéseket, javaslatokat, alternatív megoldásokat adunk, amelyeket majd a dokumentum szerzője elbírál, felhasznál.

Ha többen dolgoznak a szövegen, akkor ez különböző színekkel, a buborék szerzői feliratával megkülönböztethető. A megjegyzések megjelenítésére is több beállítás áll rendelkezésre. A dokumentum megjelenítésétől függően buborékban láthatjuk a beszúrt megjegyzéseket, vagy rá kell mutatnunk a szövegre ahhoz, hogy megjelenjen a megjegyzés.

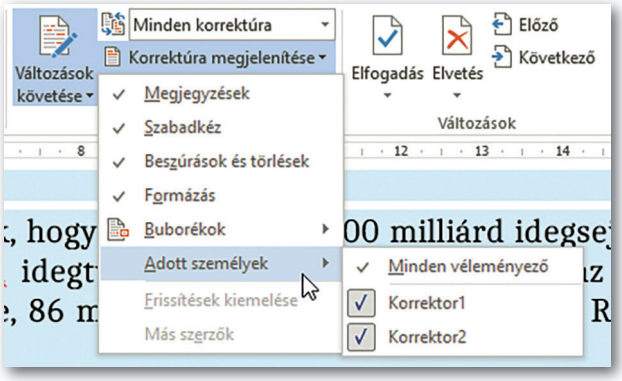


► Korrektúramegjegyzés a margón (Microsoft Word)

2. példa: A korrektúra és a megjegyzések használata

Nyissuk meg a tankönyv weboldaláról letöltött `ideg_2pelda_forras.docx` fájlt! A dokumentum a megoldott 1. példa további módosításához készített korrektúrát tartalmazza. A korrektúrát ketten (*Korrektor1* és *Korrektor2*) végezték, nevük megjelenik a megjegyzésbuborékokban. Folytassuk a dokumentum szerkesztését, javítását a megjegyzések és a korrektúra alapján!

A megjegyzésekben mindig olyan tanácsot, utasítást adunk és kapunk, amely összetett, nem automatizálható változtatást igényel. A feladat elvégzése, annak módosítása, visszautasítása esetén a megjegyzést töröljük, vagy a buborékban választ adunk.



► Korrektúra eszköztár (Microsoft Word)

A korrektúrában lévő tartalmi és formai változtatásokat például az *Elfogadás* vagy az *Elvetés* gombbal kezelhetjük. A javaslatokat változtatás nélkül végignézhetjük az *Előző* és *Következő* gombok használatával.

Feladatok

1. Munkánkat mentsük `ideg_2pelda_megoldas` néven a szövegszerkesztő alapértelmezett formátumában!
2. *Korrektor2* sok betűformázás-változtatást állított be. Ezek elsősorban a szövegben található szakkifejezések és fogalmak kiemelését, hangsúlyozását célozzák. Fogadjuk el a korrektúrában a formázási javaslatokat!
3. *Korrektor1* egyik megjegyzése arra hívja fel a figyelmet, hogy a tanulmányban több helyen szerepel a Na^+-ion és K^+-ion képlete, de anélkül, hogy a töltésük jelölése felső indexben lenne. Ezt a hibát többféle módszerrel lehet javítani; gondoljuk végig a lehetőségeket! Válasszuk ki és hajtsuk végre azt a leghatékonyabb javítási módszert, amellyel biztosan nem marad ki egyik ion képlete sem! Utána töröljük a figyelemfelhívó megjegyzést!
4. *Korrektor1* a tartalomjegyzék átnézésekor észrevette, hogy a *„Nyugalmi potenciál”* kifejezés tévesen szerepel alcímként. Ennek a bekezdésnek *Címsor 3* és nem *Címsor 2* stílusúnak kell lennie. A megjegyzésnek megfelelően javítsuk ki a stílus beállítását, majd frissítsük a tartalomjegyzéket, mert a program automatikusan nem végzi ezt el!
5. A két korrektor többi megjegyzésének megfelelően végezzük el a változtatásokat, majd töröljük a megjegyzéseket!
6. Mentsük el a szöveget a szövegszerkesztő alapértelmezett formátumában, majd készítsünk belőle egy PDF-formátumú fájlt!

Minták az 1. és 2. példához:

Az idegsejtek

1. Miért különleges az idegsejt, miben különbözik más sejttípusoktól?
2. Hányféle idegsejttípus építi fel az emberi agyat?
3. Hogyan és miben segítenek a gliasejtek az idegsejteknek?

Az idegsejtek száma

Idegsejtek, illetve azok nyúlványai az emberi szervezet szinte minden ré
nak. A legtöbb az agyban található (újabb számítások szerint 86 milliárd)
a gerincvelőben és a bélben az idegsejtek száma közel megegyezik, így
külön „agyként" is említik. Az ecetmuslicának 100–300 ezer, az egérnek
kának 1,2 milliárd, kutyáknak 2,2 milliárd, a csimpánznak 28 milliárd
milliárd idegsejtjéről számolnak be a szakirodalomban.

Válaszolj a kérdésre!

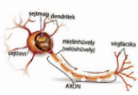
Egy magazin címlapján a következő kérdés szerepelt: Az el
a legtöbb neuronnal. Miért nem a legokosabb állatok?

Gondolkodj kritikusan!

- Fél évszázadon át azt gondolták, hogy az emberi ag
tet tartalmaz. Suzana Herculano-Houzel idegtudós
ki az agysejtek számlálására, ennek eredménye, 86
magyarázat lény

Az idegsejt felépítése

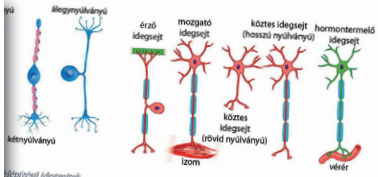
Az idegsejtek nyúlványos sejtek, rövid nyúlványaik a **dendritek**, a leghosszabb nyúlványuk az **axon**. Az információ felvétele általában a dendriteknél zajlik, a sejttestben van a legtöbb sejtalkotó, itt történik a sejtműködések irányítása és a működés szempontjából fontos anyagok előállítása (1. ábra). Az axon, illetve végződése, az általában elágazó **végfácska** köti össze az idegsejtet egy másik sejttel. Az axon sejthártyája **elektromos jel** (akciós potenciál) vezetésére specializálódott, aminek hatására az axonvégződésből **kémiai anyagok** szabadulnak fel.



1. Az idegsejtek általános felépítése

Idegsejtek típusai

Az idegsejtek attól függően, hogy a sejttestből hány nyúlvány ered, lehetnek **soknyúlványúak** (multipoláris), **egynyúlványúak** (unipoláris), **álegynyúlványúak** (pszeudounipoláris), **kétnyúlványúak** (bipoláris) (2. ábra). Működésük alapján **érzőidegsejt** (szenzoros neuron), **mozgatóidegsejt** (motoros neuron) és **közti idegsejt** (interneuron) különböztethető meg.



3. Különböző funkciójú idegsejtek

uron főként gerinctelenekre (pl. rovarok) jellemző. Gerincesekben az
ulványú idegsejtnek kezdetben két nyúlványa van, amelyek a sejttest
dnak. A soknyúlványú idegsejtek főként interneuronok és mozgató-
euronok axonja lehet rövid, és ekkor helyi kapcsolatokat alakít ki, vagy
abbi célsejtekkel létesít kapcsolatot (3. ábra).

ess!

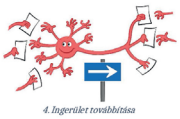
z interneuronok száma a legtöbb az emberi agyban?

Miért különleges sejttípus az idegsejt?

Az idegsejt ingerlékeny sejttípus, melynek fő feladatai az **információ** (inger) **felvétele**, **feldolgozása** (jelátalakítás, inger ingerületté alakítása), és továbbítása (4. ábra).

Az **inger** az élő szervezetet érő, annak valamilyen választ keltő hatás.

Az **ingerület**: a sejt inger hatására bekövetkező változása. Megváltozhat pl. a sejt alakja (izom), elektromos állapota (akciós potenciál alakul ki), anyagcseréje (enzimek aktiválódása révén lebontja a májsejt a glikogént glükózzá).

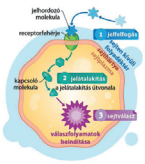


4. Ingerület továbbítása

Jelnek nevezünk mindent, ami mérhető és információt hordoz, pl. fizikai, kémiai, biológiai mennyiségek vagy azok változásai.

A **receptor** (jelfogó és jelátalakító) fogalmát különböző szerveződési szinteken is értelmezhetjük.

A receptorok a sejt felszínén vagy a sejt belsejében elhelyezkedő **fehérjemolekulák**, amelyek egy adott anyag specifikus megkötésére képesek, ez a kötés különböző válaszfolyamatok beindulását eredményezi (5. ábra). Például ha a szívizomsejtekhez futó egyik idegvégződésből noradrenalin (jel) szabadul fel és hozzákötődik a szívizomsejtek receptoraihoz (jelfogó), ennek hatására olyan folyamatok (jelátalakítás, jelerősítés) indulnak be, amelyek eredményeként a szívizomsejtek és így a szív összehúzódásainak száma és ereje is nő (vá-



5. A jelfeldolgozás sejten belüli folyamata

lasz).

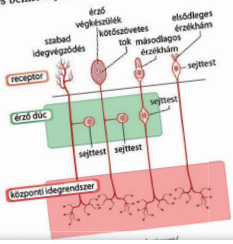
Sejtszinten receptornak számítanak az **érzékelősejtek** (speciális idegsejtek, módosult hámsejtek). A receptorsejt a külső és belső környezet fizikai (fény, hő, mechanikai) és kémiai (szag, íz) ingereit alakítja át elektromos jellé. Ahhoz, hogy az információkat a szervezet feldolgozza, le kell fordítani ezeket az idegsejtek nyelvére, amit **jelátalakításnak** nevezünk. Például a hangok, mint mechanikai ingerek a fülünkben elektromos jelekké alakulnak, ami több lépésben majd az agyba jut, ahol is megtörténik az elektromos jel értelmezése.

Fedezd fel!
Keress a környezetedben olyan berendezéseket, amelyek jelátalakítót tartalmaznak! Mely jeleket alakítanak egymásba?

Legyél narrátor!
Keress egy jelátviteli folyamatokról szóló animációt, és a hang levételével próbáld meg elmondani a saját szavaiddal, hogy mi történik a jelátvitel során!

A receptorsejtek típusai

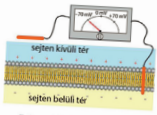
Felépítésük alapján egyes érzőidegsejtek lehetnek szabad idegvégződésűek, m
ződését kötőszövetes tok és benne sejtek veszik körül (idegvégkészülékek) (



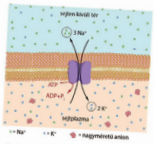
Az idegsejtek működése

Nyugalmi potenciál

Ha egy nyugalomban lévő, idegsejt belsejébe és a sejten kívüli terébe is elhelyezünk egy érzékelőelektródát, akkor a sejten belüli és a sejten kívüli tér között potenciálkülönbséget (feszültséget) mérhetünk, amit nyugalmi potenciálnak nevezzük (7. ábra). Értéke általában 0 és –100 mV közötti (jellemzően –70 mV).



7. A nyugalmi potenciál mérése



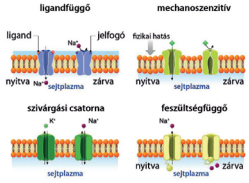
8. A nyugalmi potenciált kialakító Na^+/K^+-pumpafehérje

…ően a sejt belső és külső tere közötti eltérő ionösszetételre vezethető
…üli térben például a káliumionok, míg a sejten kívüli térben a nátriumio-
…nagyobb koncentrációban. Ezt a sejthártyában lévő fehérje, a Na^+/K^+-
… működése okozza (8. ábra).
…nokon kívül a sejt belsejében a negatív töltésű fehérjék, míg a sejt külső
…k találhatók nagyobb koncentrációban.

…djatok!
…el a működéséhez ATP-t a Na^+/K^+-pumpa?

Mi következik abból, hogy a Na^+/K^+-pumpa mindig működik? Nyilvánvaló, hogy egy idő után a pumpa az összes Na^+-iont kipumpálná, ha nem térne vissza állandóan a Na^+-ion a sejtbe.

Kiderült, hogy sejthártyában ioncsatornák vannak. Ezek egyik típusa olyan, amely állandóan kismértében nyitva van. A káliumionokat kiengedő csatornák nagyobb áteresztőképességűek, mint a nátriumionokat beengedők (szivárgási csatornák). Vannak más típusú ioncsatornák is, melyek, akkor nyílnak meg, ha egy kémiai anyag (ligand) az ioncsatornához kapcsolt receptorhoz kötődik. Mások meghatározott feszültségértékeknél, vagy mechanikai hatásra nyílnak meg.



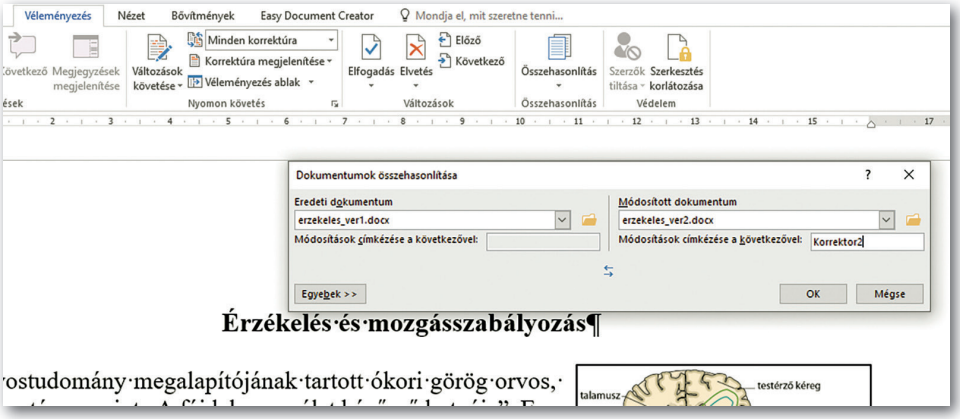
Megjegyzések

- A K^+-ionok mintegy 20-szor könnyebben jutnak át a membránon nyugalomban, mint a Na^+-ionok, ezért a nyugalmi potenciál kialakításában ezek az ionok a főszereplők.
- Már igen kevés ion átlépése jelentős feszültségkülönbség kialakulását eredményezi a membrán két oldala között.

Tartalom:

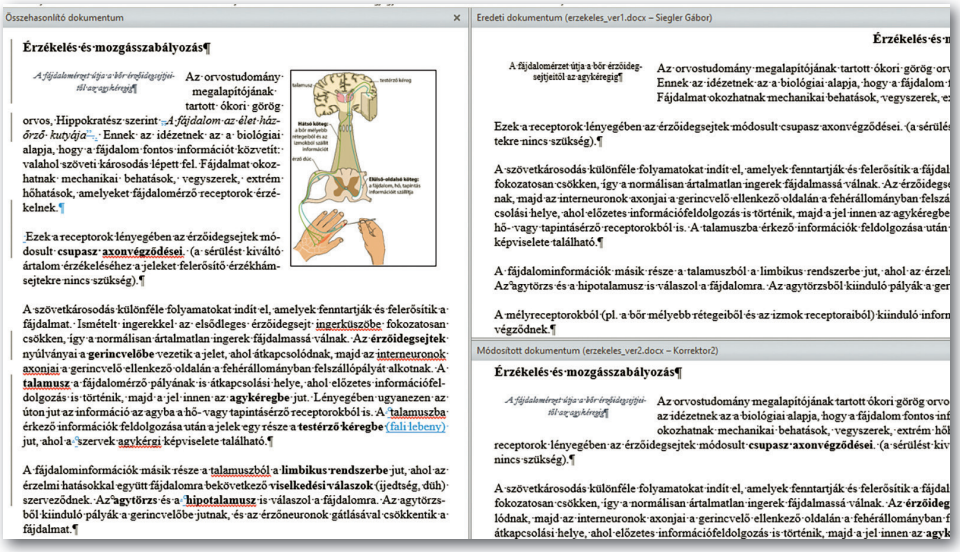
Dokumentumok összehasonlítása

Ha egy dokumentum szerkesztése során a *Változások követése* funkció nem volt bekapcsolva, akkor a dokumentum változásait úgy tudjuk felderíteni, hogy összehasonlítjuk a mentett verziókat. Az összehasonlítás alapján az eltéréseket egy új dokumentumban jeleníti meg a szövegszerkesztő program korrektúraként. Ebben elfogadhatjuk vagy elutasíthatjuk a módosításokat.



▶ Két összehasonlítandó dokumentum megadása (Microsoft Word)

A dokumentumok összehasonlítását például a *Véleményezés* > *Összehasonlítás* gombbal (illetve LibreOffice Writerben a *Szerkesztés* > *Változások követése* > *Dokumentumok összehasonlítása* menüponttal) indíthatjuk. Az eredeti és a módosított dokumentum összehasonlítása egy új ablakban jelenik meg. Az eltérések korrektúrában, illetve a bal szélen fekete vonallal vannak jelölve.



▶ Dokumentumverziók összehasonlítása (Microsoft Word)

A különbségek hivatkozásai általában újabb ablakban jelennek meg, ahol egyenként dönthetünk az elfogadásukról vagy elvetésükről. Ha az ablak nem jelenik meg, akkor azt például a *Véleményezés* ablak gombjának bekapcsolásával (illetve a *Szerkesztés* > *Változások követése* > *Megjelenítés* menüponttal) nyithatjuk meg. A korrektúrák kezelése után új állományba menthető a dokumentum az előző verziók megváltoztatása nélkül.

3. példa: Dokumentumok összehasonlítása

Nyissuk meg a letöltött `erzekeles_ver1.docx` és `erzekeles_ver2.docx` fájlt! A dokumentumok tartalma a forrásként már használt biológiakönyvből származik, témájuk az érzékelés idegfolyamatai. A két állomány szövege majdnem teljesen azonos, de csak majdnem, mert *Korrektor1* és *Korrektor2* az eltérő formázások mellett néhány szót megváltoztatott.

Feladatok

1. Nyissuk meg a két állományt a szövegszerkesztő program dokumentumok összehasonlításának lehetőségével!
2. A megfelelő funkció bekapcsolásával határozzuk meg, hogy hány különbség van a két verzió között!
3. Az *Elfogadás* és *Elvetés* gomb segítségével saját belátásunk szerint fogadjuk el vagy hagyjuk figyelmen kívül a korrektúra javaslatait!
4. Az összes korrektúra kezelése után munkánkat mentsük el `erzekeles_kesz` fájlnéven!

Online szövegszerkesztés

Az interneten elérhető szövegszerkesztő alkalmazások többféle módon is csoportosíthatók. Az egyik fontos szempont a felhasználói kör nagysága, ennek alapján léteznek egy- és többfelhasználós programok. Az előző csoportba tartozó alkalmazások lényegében nem tartalmaznak több funkciót az operációs rendszer részét képező editorokhoz képest. Az egyszerűbbeknél még a használt böngészőprogram típusa sem mindegy.

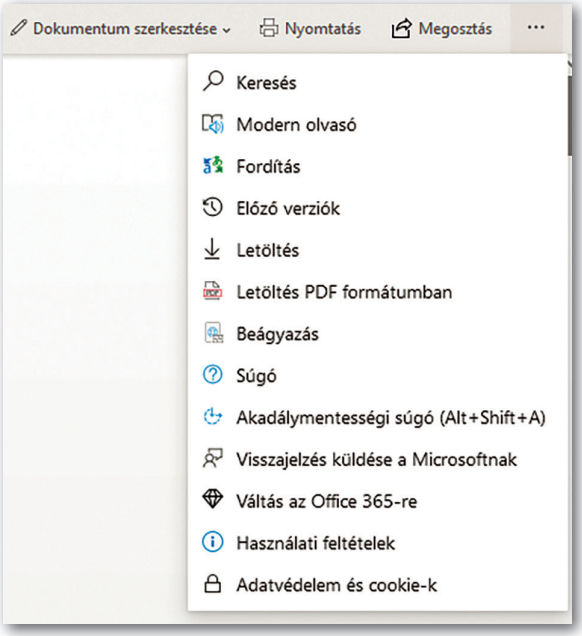
A **csoportmunkához** (kollaboratív munka) jól használható többfelhasználós, **felhőalapú alkalmazások** tudása, funkcionalitása folyamatosan növekszik. A számuk is egyre több, de néhány példa a teljesség igénye nélkül: Google Dokumentumok (Google Docs), Microsoft Word Online, az iCloud oldalai (Pages for iCloud). Mindegyiknek van ingyenes, magánszemélyeknek fejlesztett és vállalati, professzionális változata. A változatok egymástól nem egyszerűen a felhasználók számában, esetleg a jogosultsági beállítások kezelésében, hanem a többi szoftverrel való **integráltságukban** térnek el. Például a **verziókövetés,** az adatok fogadása és továbbítása a vállalati alkalmazásban már nélkülözhetetlen.

Nemcsak dokumentumállományok tárolása történik az adatfelhőben, hanem azoknak a szerkesztése is böngészőprogrammal végezhető el.

Ha többen szerkesztenek egy közös dokumentumot, akkor már a konkurencia problémája lép fel. Ezt a hálózati megoldásoknak úgy kell kezelniük, hogy ne történhessen adatvesztés. Minden szerkesztőnek folyamatosan a dokumentum aktuális állapotát kell látnia, és a többiek munkájáról információkat kapnia. A csoportmunka egyik leglényegesebb kérdése a **megosztás** minél finomabb beállítása.

► Szövegszerkesztés a Google Dokumentumokkal

Az online rendszerek bármilyen platformról (számítógépről, okostelefonról, mobileszközökről) elérhetők, bár némelyikre **külön alkalmazást** kell telepíteni.



▶ A Microsoft Word Online funkciói

A kijelző mérete miatt mobileszközökön nehéz szerkeszteni a szöveget, de egy-két szót beírhatunk és javíthatunk. Praktikusan inkább csak a megosztott dokumentumok olvasására használjuk az applikációt.

Feladatok

Készítsünk csoportmunkában a kémiaórán tanult legfontosabb vegyületek összefoglalásához, ismétléséhez anyagismereti kártyákat!

1. Az anyagismereti kártyák egységes megjelenítését közös sablon használatával tervezzük és valósítsuk meg! A sablont mentsük el `kartyasablon` néven a szövegszerkesztő sablonformátumában, és osszuk meg egymás között!
2. A sablon egy általános szerkezetű kitöltendő táblázatot tartalmazzon, amely egységes sorrendben és stílussal tartalmazza a szempontoknak megfelelő cellacímeket!
3. A sablonban hozzunk létre és alkalmazzunk néhány stílust a megfelelő bekezdésekre, illetve szövegrészekre, például: *cím, cellacím és mértékegység!*
4. A munkát, vagyis a vegyületek anyagismereti kártyáinak elkészítését osszuk be előre! Emlékeztetőül néhány fontosabb vegyület a szerves kémiai tanulmányokból: metán, etán, propán, etén, propén, vinil-klorid, buta-1,3-dién, izoprén stb.
5. A kártyák elkészítése a `kartyasablon` sablon alapján történjen! Forrásként a kémiatankönyv vagy megbízható internetes oldalak használhatók.
6. A munkában részt vevők előre állapodjanak meg abban, hogy ki kinek az elkészült anyagismereti kártyáját fogja ellenőrizni és korrektúrával módosítani, illetve megjegyzésekkel ellátni. Többszöri egymás közötti cserével és javítással az elkészített dokumentumok megbízhatósága nő. A változtatásokat, javaslatokat mindig az eredeti szerző hagyja jóvá!

Minta az Anyagismereti kártyák feladathoz:

Anyagismereti kártya

Szabályos (IUPAC) név: **Egyéb nevek:**		**Kép:**
Összegképlet:	**Szerkezeti képlet:**	**Olvadáspont (°C):**
		Forráspont (°C):
Színe:	**Szaga:**	**Halmazállapota (25 °C, 0,1 MPa):**
Oldhatósága vízben:		**Sűrűsége (g/cm³):**
Előfordulása a természetben:		
Előállítása:		
Felhasználása:		
Élettani hatása:		
Egyéb:		

Anyagismereti kártya

Szabályos (IUPAC) név: *klór-etén* **Egyéb nevek:** *vinil-klorid, klór-etilén*		**Kép:** 
Összegképlet: C_2H_3Cl	**Szerkezeti képlet:** $H_2C{=}CHCl$	**Olvadáspont (°C):** *-153,7*
		Forráspont (°C): *-13,4*
Színe: *színtelen*	**Szaga:** *szagtalan, nagyobb koncentrációban édeskés*	**Halmazállapota (25 °C, 0,1 MPa):** gáz
Oldhatósága vízben: *oldhatatlan*		**Sűrűsége (g/cm³):** 0,970 (forrásponton)
Előfordulása a természetben: nem fordul elő		
Előállítása: $CH_2Cl{-}CHCl \rightarrow H_2C{=}CH{-}Cl$ $CH{\equiv}CH + HCl \rightarrow H_2C{=}CH{-}Cl$ higany(II)-klorid katalizátor jelenlétében		
Felhasználása: A vinil-klorid polimerizációjával PVC-t (poli-(vinil-klorid)) állítanak elő: fóliák, csövek, padlóburkolat.		
Élettani hatása: mérgező gáz		
Egyéb: Forrás: *https://hu.wikipedia.org/wiki/Vinil-klorid* Utolsó letöltés: 2021. 07. 28.		

► A `kartyasablon` sablon szerkezete és a vinil-klorid anyagismereti kártyája

D Táblázatkezelés

Az elmúlt évek során már megismerkedtünk a táblázatkezelő programok használatával; ebben a fejezetben tovább folytatjuk ezeket a tanulmányokat. Az első leckében felidézzük és rendszerezzük azokat az elméleti ismereteket, amelyekre szinte minden esetben szükségünk volt, majd további problémák vizsgálatával bővítjük tudásunkat. A fejezet második részében pedig megismerkedünk azokkal az eszközökkel, amelyeket a táblázatkezelő programok nagy adathalmazok kezelésére kínálnak.

Az alapismeretek áttekintése

Szám, szöveg, logikai típusok

A táblázatkezelők celláiban háromféle adat szerepelhet: **szám, szöveg** vagy **logikai érték.** A táblázatkezelők ezeket a típusokat alapértelmezés szerint a cella tartalmának igazításával is megjelenítik: a számokat jobbra zárják, a szöveges adatot balra, a logikai értékeket pedig középre igazítják.

1. példa: Ösztöndíjpályázat

Alábbi példánkban az Irka Iskola alapítványának ösztöndíjpályázatára az aktuális tanévben benyújtott pályázatok összesítését látjuk. Az egyes oszlopok tartalmazzák a tanuló nevét és osztályát; születési dátumát; előző tanév végi átlagát; azt, hogy eddig hány óra közösségi szolgálatot teljesített (X); nyert-e már korábban a pályázaton; most milyen arányban támogatta a kuratórium, illetve hogy ezúttal mennyi ösztöndíjat kapott. (Ha az utolsó oszlop értéke üres, akkor a kuratórium nem támogatta a pályázatot.)

	A	B	C	D	E	F	G	H
1	**Név**	**Osztály**	**Született**	**Átlag**	**X**	**Korábban**	**Támogatás**	**Összeg**
2	Albert László	12.b	2002.12.11	4,10	49	HAMIS	82%	80 000 Ft
3	Címer Csaba	9.c	2006.08.31	3,67	4	HAMIS	23%	
4	Felsőrákosi Richá	10.a	2004.03.31	4,10	22	HAMIS	76%	80 000 Ft
5	Kovács Hedvig An	10.c	2004.06.23	4,66	12	IGAZ	78%	80 000 Ft
6	Nagy Irma	10.b	2002.07.01	4,50	44	HAMIS	98%	80 000 Ft
7	Tóth Hanga	11.a	2003.08.12	4,80	50	IGAZ	80%	80 000 Ft
8	Zémann Dóra	11.b	2003.06.07	3,14	3	IGAZ	31%	

► A benyújtott pályázatok összesítése. A táblázatkezelők a szöveges adatokat balra, a számokat jobbra, a logikai értékeket középre zárják

Az *A* és *B* oszlopban **szöveges adatokat** látunk. Szöveg bevitele esetén gyakori probléma, hogy a szöveg nem fér ki egy cellába. Ilyenkor bevett megoldás, hogy szélesebbre állítjuk az oszlopot, vagy több sorba tördeljük a cella tartalmát a *Sortöréssel több sorba* (illetve a *Szöveg tördelése*) gomb segítségével.

Az *F2:F8* tartomány celláiban **logikai értékek** szerepelnek. A logikai érték csak kétféle lehet: IGAZ vagy HAMIS. A táblázatkezelő programok a logikai kifejezéseket is ki tudják értékelni, így például az =(5>3) képlet eredménye IGAZ.

A *C2:E8* és a *G2:H8* tartomány cellái **számokat** tartalmaznak. Ha egy szám nem fér ki a cellába, akkor abban a ###### hibaüzenet jelenik meg. A táblázatkezelő programok a számokat egyformán tárolják (az Excel például 15 tizedesjegy pontossággal), azonban többféle **számformátumban** képesek megjeleníteni őket. Az alábbi ábrán a táblázat első adatsorát beállítások nélkül, „általános számformátumban" látjuk:

	A	B	C	D	E	F	G	H
1	**Név**	**Osztály**	**Született**	**Átlag**	**X**	**Korábban**	**Támogatás**	**Összeg**
2	Albert László	12.b	37601	4,1	49	HAMIS	0,82	80000

► Az előző táblázat első adatsora számformátumok beállítása nélkül

A leggyakoribb számformátumok

Szám: A legegyszerűbb számformátum, amely a számokat ezres tagolással, két tizedesjegy pontossággal jeleníti meg. Ezt a formátumot például a 000 (illetve a 0,0) paranccsal választhatjuk ki. A tizedesjegyek számát növelhetjük és csökkenthetjük például a ,0 ,00 ,00 ,0 (illetve a ,00 ,00) parancs alkalmazásával. Ezt a formátumot használtuk a *D* oszlopban, ahol a tanulók átlaga két tizedesjegy pontossággal jelenik meg. Ne felejtsük el, hogy a két tizedesjegy csak a megjelenítésre vonatkozik, a tárolt tizedesjegyek száma lehet ennél kevesebb vagy több is. A számok pontosságát adott számú tizedesjegyre például a **KEREKÍTÉS** függvénnyel módosíthatjuk.

Az alábbi példában a harmadik sor szemlélteti, hogy a második sorban hogyan módosítottuk az első sor adatait. Jól látható, hogy a formátumok módosítása esetén a táblázatkezelő az eredeti értékekkel számol tovább.

	A	B	C	D	E	F	G
1	4,444	3,333	7,777		4,444	3,333	7,777
2	4,4	3,3	7,8		4,4	3,3	7,7
3	2× ,00 ,0	2× ,00 ,0	=A2+B2		=KEREKÍTÉS(E1;1)	=KEREKÍTÉS(F1;1)	=E2+F2

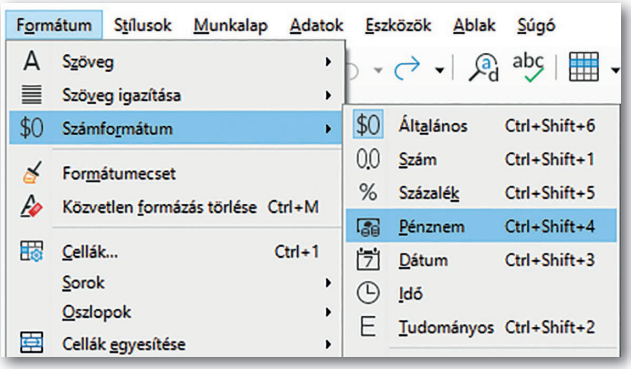
► Az adatokat az 1. sorból a 2. sorba a 3. sorban előírt módon állítottuk elő. Formázáskor a táblázatkezelő az eredeti értékekkel számol tovább

Százalék: Egy szám százalék alakja a szám százszorosa, kiegészítve a százalékjellel (%), például 0,34 = 34%. A táblázatkezelő programokba ezt az értéket kétféleképpen vihetjük be. Az egyik lehetőség, hogy beírjuk a szám százszorosát, és utána (szóköz nélkül) a százalékjelet (34%); a másik, hogy beírjuk a számot (0,34), majd a *Százalék* % gomb segítségével alakítjuk százalék formátumúvá. Ezt a formátumot alkalmaztuk a *G* oszlop celláiban.

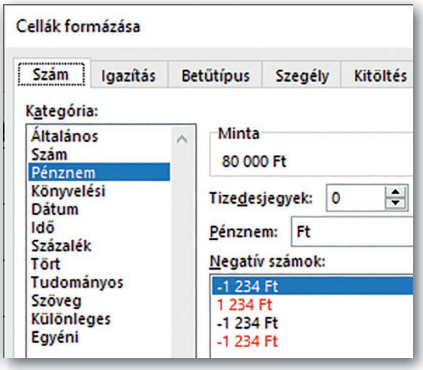
Pénznem: Ha egy számot pénznem formátumúvá alakítunk, akkor a táblázatkezelő program kiegészíti azt a pénznem jelével, illetve a beállításoktól függően ellátja ezres tagolással, esetleg eltérő színnel jeleníti meg a negatív számokat, stb.

A pénznemet beírhatjuk közvetlenül is, például: 123 Ft. Ilyenkor ügyeljünk a szabályos bevitelre, mert a szabálytalanul beírt összeget (például 123ft) a táblázatkezelő szövegként értelmezi! Van olyan táblázatkezelő program, amely külön formátumot tartalmaz arra az

esetre, ha a pénznem jelét és a tizedesvesszőt egymás alá akarjuk rendezni. (Ilyen például a Microsoft Excelben a *Könyvelési* formátum.)

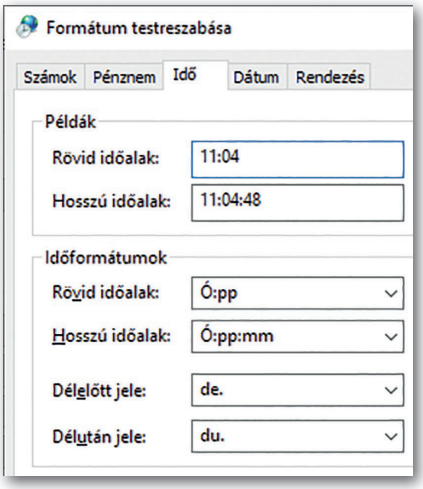


► Pénznem formátum alkalmazása (LibreOffice Calc)



► A pénznem további lehetőségei (Excel)

Dátum és idő: A táblázatkezelő programok a hétköznapi életben megszokottól (év-hónap-nap, óra-perc-másodperc) eltérő rendszert alkalmaznak. Itt *az idő egysége az egy nap*, és számozása 1900. január 1. 0:00-val indul (ez az 1,00). Ennek megfelelően például a 30,25 megfelel 1900.01.30. 06:00-nak. A rendszer nagy előnye, hogy két dátum különbsége a köztük eltelt *éjszakák* száma (ha nem tároljuk az órát és a percet). A példából az is látszik, hogy az idő a szám tizedes tört részében tárolódik, és azt fejezi ki, hogy az hányad része a napnak (például a 6:00:00 számformátum esetén 0,25).



► Az időformátum beállítása (Windows 10)

A dátumot és az időt is beírhatjuk közvetlenül (ekkor ügyeljünk annak szabályos bevitelére, például az időegységek elválasztójeleire), de nagyon sok dátum- és időformátum közül választhatunk a táblázatkezelő menürendszerében is. A pontos idő beszúrása az aktív cellába a Ctrl + Shift + . (pont) billentyűkombinációval történik. (LibreOffice Calcban pedig a Ctrl + . [pont] billentyűkkel lehet beszúrni az aktuális dátumot.)

Ahhoz, hogy helyesen alkalmazzuk a táblázatkezelőkben a dátum- és időformátumokat, illetve ki tudjuk használni az előnyeiket, a táblázatkezelő programok nagyon sok függvényt bocsátanak rendelkezésünkre.

További számformátumok: A *Tudományos számformátum* a matematikában normálalaknak nevezett formátum, csupán kicsit eltérő módon jelenik meg: $6{,}02 \cdot 10^{23}$ helyett 6,02E+23. A *Tört formátum* alkalmazásánál pedig a táblázatkezelő az adott tizedes törtet az általunk megadott pontossággal közönséges tört alakban jeleníti meg.

Fontos tudnunk, hogy a táblázatkezelő programok a tizedes tört elválasztójelét (például a tizedesvesszőt), a pénznem jelét (például a Ft-ot), a dátum formátumát (például 2021.03.15) stb. az operációs rendszer területi beállításaiból veszik.

Cellahivatkozások, képletek

Ha a cella tartalma képlet, akkor a cellában annak értékét, a szerkesztőlécen pedig magát a képletet látjuk. Ha a cellákban a képletet szeretnénk látni, akkor arra a Microsoft Excelben az ALT + . (pont) billentyűkombinációval válthatunk át. Képlet alkalmazása esetén gyakori a `#NÉV?` hibaüzenet, amely egy függvény nevének elírására utal.

A képletben lévő cellahivatkozást **relatív cellahivatkozásnak** nevezzük, ha az a képlet másolásakor a másolás irányának megfelelően módosul. Ilyenkor a táblázatkezelő program nem a ténylegesen hivatkozott cellacímet tárolja, hanem annak az aktuális cellához viszonyított helyzetét.

Abszolút cellahivatkozás esetén a cella címe másoláskor nem változik, ekkor a táblázatkezelő program a cella tényleges helyét tárolja.

Az alábbi ábrán látható példa esetén a nettó árat a mennyiség és az egységár szorzatából kapjuk: `D3=B3*C3`, a bruttó ár ennek az áfával megnövelt értéke: `E3=D3+D3*$E$1`.

	A	B	C	D	E
1				Áfa:	27%
2	Termék	mennyiség (kg vagy l)	nettó egységár	nettó ár	bruttó ár
3	Glettanyag	120	300 Ft	36 000 Ft	45 720 Ft
4	Falfesték	140	350 Ft	49 000 Ft	62 230 Ft
5	Mestertapasz	4	3 600 Ft	14 400 Ft	18 288 Ft
6	Alapozó festék	4	1 700 Ft	6 800 Ft	8 636 Ft
7	Zománcfesték	4	3 300 Ft	13 200 Ft	16 764 Ft

▶ Példa relatív és abszolút cellahivatkozásra: D3-as cella: `=B3*C3`, E3-as cella: `=D3+D3*$E$1`

	A	B	C	D	E	F
1	Munka	Alapter. (m^2)	Anyag (m^2)	Munka (m^2)	Anyag (teljes)	Munkadíj (teljes)
2	Alapozás	100	200 Ft	600 Ft	20 000 Ft	60 000 Ft
3	Glettelés	100	400 Ft	1 000 Ft	40 000 Ft	100 000 Ft
4	Festés	100	500 Ft	1 000 Ft	50 000 Ft	100 000 Ft
5	Csiszolás, tapaszolás	20	800 Ft	3 000 Ft	16 000 Ft	60 000 Ft
6	Mázolás alapozóval	20	400 Ft	1 000 Ft	8 000 Ft	20 000 Ft
7	Mázolás zománccal	20	500 Ft	1 000 Ft	10 000 Ft	20 000 Ft

▶ Példa vegyes és relatív cellahivatkozásra: E2-es cella: `=$B2*C2`

A **vegyes cellahivatkozásban** az egyik koordináta abszolút, a másik relatív: például *D$3* vagy *$E4*. Másoláskor az egyik koordináta nem változik (*$3* vagy *$E*), a másik viszont igen (*D* vagy *4*). Tipikus felhasználási területe, ha ugyanazt az adatot két különböző értékhez szeretnénk hozzárendelni.

Az ábrán látható példa esetén az alapterületet az 1 m^2-re eső anyagköltséggel, majd az 1 m^2-re eső munkadíjjal szorozzuk az *E* és *F* oszlopokban, így az *E2*-es cellában az `=$B2*C2` szerepel, amely másolással „lefelé” és „jobbra” is helyes marad.

Feladatok

1. Mint a kilencedikes digitáliskultúra-tankönyvünk egyik szövegszerkesztés példájában olvashattuk: „Az idő pénz.” (Kitől származik ez az idézet?) Írjuk be a mai dátumot, és alakítsuk át pénz formátumúvá! Hány forintot „ér” a mai nap a táblázatkezelő szerint?
2. A számológépek megjelenése előtt a bonyolultabb számítások eredményét táblázatokból keresték ki. Az alábbi példa egy négyzetgyöktáblázat részletét mutatja. Készítsük el vegyes cellahivatkozás alkalmazásával! (Az *A* oszlopban a szám egészrésze, az első sorban a törtrésze szerepel.)

	A	B	C	D	E	F	G	H	I	J	K
1		0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
2	0	0,0000	0,3162	0,4472	0,5477	0,6325	0,7071	0,7746	0,8367	0,8944	0,9487
3	1	1,0000	1,0488	1,0954	1,1402	1,1832	1,2247	1,2649	1,3038	1,3416	1,3784
4	2	1,4142	1,4491	1,4832	1,5166	1,5492	1,5811	1,6125	1,6432	1,6733	1,7029

Dátum és idő. Szöveges adatok (ismétlés, kiegészítés)

2. példa: Nyári tábor

Az alábbi táblázat az Irka Gimnázium nyári táborainak az adatait tartalmazza. Az adatokat olvassuk be, vagy a vágólapon át illesszük be a táblázatkezelő munkalapjára a könyv weboldaláról letöltött `taborok.txt` fájlnevű, tabulátorokkal tagolt szöveges állományból! A dátumformátumot a mintának megfelelően alakítsuk ki! Szúrjuk be a megfelelő oszlopokat, és formázzuk meg a táblázatot a mintának megfelelően!

	A	B	C	D	E	F	G	H	I	J
1	**Kód**	**Felelős**	**Helyszín**	**Típus**	**Kezdete**		**Vége**		**Napok**	**Létszám**
2		Szabó Ingeborg	Soroksár	némettábor	június 17.		június 19.			24
3		Koczka Richárd	Fonyód	matektábor	június 18.		június 22.			24
4		Csontos Csaba	Sülysáp	kézművestábor	június 21.		június 22.			32
5		Farkas Fábián	Zamárdi	csapatépítés	június 23.		június 25.			32
6		Fekete Lilla	Balaton	biciklitúra	június 23.		június 25.			24
7		Katona Örs	Rezi	sziklamászás	június 25.		június 30.			16

▸ Az Irka Gimnázium nyári táborai. A *Kód* mezőbe a tábor azonosítója kerül; az *F* és *H* oszlopokba a hét adott napjának megnevezése; a *Napok* oszlopba az, hogy hány napig tart a tábor; végül a *Létszám* oszlop tartalmazza a résztvevők maximális létszámát

A *Napok* oszlopban határozzuk meg, hogy hány naposak az egyes táborok! Két dátum különbsége a köztük eltelt éjszakák száma, így az *I2*-es cellában a napok számát például az `=G2-F2+1` képlettel határozhatjuk meg.

Adjuk meg az *F,* illetve a *H* oszlopban, hogy a tábor kezdete, illetve vége a hét melyik napjára esik! Erre a feladatra több megoldással is találkoztunk, például:

- A **SZÖVEG** függvény a dátumból közvetlenül megadja a hónap vagy a hét napjának a nevét. A paraméterezés a táblázatkezelő programtól függően eltérhet, például az *F2*-es cellában Microsoft Excel esetén: `=SZÖVEG(E2;"nnnn")`, LibreOffice Calc esetén pedig: `=SZÖVEG(E2;"nnn")`. Az eredmény szöveges adat lesz.
- Alkalmazhatunk egyéni formátumot is. Ekkor például a *H2*-es cellába az `=G2` másolható képlet kerül, míg a *H* oszlopban alkalmazott egyéni formátum: `"nnnn"` (Microsoft Excel), illetve `"nnn"` (LibreOffice Calc).
- Használhatjuk a **HÉT.NAPJA** függvényt, amely a nap sorszámát adja vissza (a vasárnap az 1-es). A sorszámból pedig egy segédtáblázattal, keresőfüggvény segítségével (például **FKERES**) kereshetjük vissza az adott sorszámú nap nevét.

3. példa: Vásárlási kedvezmény

Egy áruházlánc annyi százalék kedvezményt ad a vásárlóknak, ahányadik hónapban születtek. A kedvezményt azonban csak akkor vehetik igénybe, ha a vásárlás összege forintban kifejezve meghaladja a születési évüket. A kedvezmény igénybevételéhez a vásárlónak meg kell adnia a születési idejét. Az adatok egy részletét a következő ábrán láthatjuk. Határozzuk meg a kedvezmény összegét!

	A	B	C	D	E
1	**Vásárlás**	**Születési idő**	**Év**	**Hónap**	**Kedvezmény**
2	11 200 Ft	1988.12.28	1988	12	1 344 Ft
3	1 400 Ft	2002.03.14	2002	3	
4	5 600 Ft		1900	1	
5	4 760 Ft	2006.10.01	2006	10	476 Ft
6	8 888 Ft	2008.08.08	2008	8	711 Ft

► A vásárlási kedvezmény a születési évtől és hónaptól függ, ezeket a *C* és *D* oszlopban állítjuk elő

A születési időt dátumként kezeljük, amely ténylegesen az 1900. 01. 01. óta eltelt idő napokban kifejezve, így ez a 2020-as évben egy 40 000 és 50 000 közé eső szám. Ebből a születés évét többféleképpen is meghatározhatjuk:

- Alkalmazhatjuk ezúttal is a **SZÖVEG** függvényt. Sajnos ekkor az évszámot szövegként kapjuk, amellyel csak akkor tudunk tovább dolgozni, ha számmá alakítjuk. (Ezt például az **INT** függvénnyel tehetjük meg.)
- Egy másik megoldás lehet, hogy a napok számát elosztjuk 365-tel. Ekkor azonban ügyelnünk kell a szökőévekre is.
- Egy újabb lehetőség, hogy a táblázatkezelők beépített függvényeit alkalmazzuk: egy időpontból a megfelelő értékeket az **ÉV, HÓNAP, NAP, ÓRA, PERCEK,** valamint az **MPERC** függvények adják vissza számként.

Mintapéldánkban így a *C2*-es cellába például az `=INT(SZÖVEG(B2;"éééé"))`, míg a *D2*-es cellába például az `=HÓNAP(B2)` képlet kerülhet. (LibreOffice Calc esetén `"éééé"` helyett `"yyyy"`-t kell írnunk.) Végül az *E2*-es cellában a kedvezmény mértékét például a következő képlettel határozhatjuk meg:

```
=HA(ÉS(B2<>"";A2>C2);A2*D2/100;"")
```

Hasonló módon, az év, hónap és nap sorszámából a dátumot például a **DÁTUM** függvény, míg az órából, percből és másodpercből az időt például az **IDŐ** függvény adja vissza.

4. példa: Szöveges adatok kezelése

A nyári táborok adatait a szervezők közzéteszik az iskola weblapján. Az iskola dinamikus webet használ, az adatok pedig egy adatbázisba kerülnek, amely lehetővé teszi a tanulók online jelentkezését. Az adatbázisban az egyes táborok azonosításához a rendszergazda egy kódot generál, amely a tanár vezeték- és utónevének első két betűjéből áll, végig csupa kisbetűvel. Készítsük el ezt a kódot az *A* oszlopban! (A megoldás során érdemes segédoszlopokat használnunk.)

Monogramkészítéssel már a kilencedik évfolyamon is találkoztunk. Ott ehhez a következő szövegkezelő függvényeket alkalmaztuk: **BAL** (a vezetéknév első két betűjéhez), **SZÖVEG.KERES** (a szóköz helyének megkereséséhez), **KÖZÉP** (a szóköztől számított két betű kivágásához), **JOBB** (a szöveg jobb oldalán lévő karakterek visszaadásához). Így esetünkben például a *Q2*-es segédcellába a következő képletet írhatjuk:

```
=BAL(B2;2)&KÖZÉP(B2;SZÖVEG.KERES(" ";B2)+1;2)
```

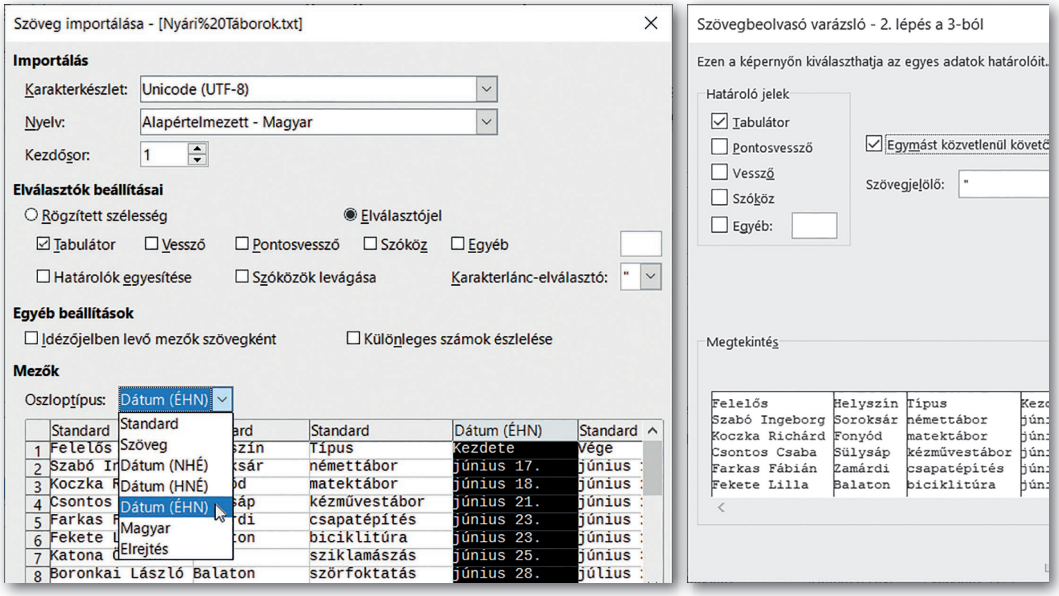
Ez a képlet még vegyesen használja a kis- és nagybetűket. A szöveget kisbetűssé például a **KISBETŰ,** végig csupa nagybetűssé pedig a **NAGYBETŰS** függvénnyel alakíthatjuk. Az *A2*-es cellába tehát a következő képletet írhatjuk: `=KISBETŰ(Q2)`.

Szöveges adatot tartalmazó oszlopok kezelésénél gyakran gondot okozhatnak a fölösleges szóközök (például a dupla szóköz a vezeték- és utónév között vagy a „láthatatlan”, fölösleges szóköz a szöveg végén). A fölösleges szóközöket a szövegből a **KIMETSZ** függvény távolítja el. (A függvény a szöveg „belsejében” is csak egy-egy szóközt hagy meg.)

Adatok importálása, beillesztése

A táblázatkezelő programok a saját formátumukon kívül többnyire más formátumokat is alapértelmezetten kezelnek. Például a Microsoft Excel és a LibreOffice Calc egymás fájljait általában minimális hibával be tudják olvasni, illetve hasonlóan jól kezelik a tabulátorokkal *(txt)* vagy a vesszővel *(csv)* tagolt szöveges állományokat is.

Az alapértelmezett formátumoktól eltérő esetekben a táblázatkezelők általában „varázslókkal” támogatják az adatok beolvasását. Ezek a segédprogramok többnyire automatikusan elindulnak a fájl megnyitásakor. Bár a beolvasás menete programonként eltérő, mindig megadhatjuk a fájl kódolását, a cellákat elválasztó jeleket, illetve segítséget kapunk a különleges adatok (például dátum) beolvasásához is.



▶ Szöveg importálását a LibreOffice Calc (balra) egy lépésben, a Microsoft Excel (jobbra) három lépésben végzi el; az ábrán a 2. lépést látjuk

Az adatokat legkényelmesebben a vágólapon át másolhatjuk be. Az adatok vágólapon át történő bemásolásakor – az adatok származási helyétől függően – több probléma is előfordulhat. Ezért egyes programokban (például a LibreOffice Calc) ilyenkor automatikusan elindul a szöveg importálása funkció.

Gyakori probléma, hogy tizedesvessző helyett tizedespont van, illetve hogy a számok ezres tagolására nem törhető szóközt (ALT + 0160 kódú karakter) használtak. A probléma egyszerűen javítható az adott oszlopban a karakterek cseréjével, első esetben tizedesvesszőre, második esetben üres karakterre.

A másik gyakori probléma, hogy a speciális karakterekkel (például vessző, pontosvessző, szóköz) tagolt adatok egy oszlopba kerülnek. Ezért a táblázatkezelő programok menüből elérhető parancsot tartalmaznak az adatok szétválasztására is, amelyet például az *Adatok* > *Szövegből oszlopok* (illetve az *Adatok* > *Szöveget oszlopokba*) menüponttal érhetünk el. Az elinduló „varázsló” hasonlít a szöveg importálását segítő funkcióhoz. Ezt a parancsot mintapéldánkban a vezeték- és utónév szétválasztására is használhatjuk, ha elválasztójelként a szóközt adjuk meg.

Feladatok

1. A nyári táborokat az iskola alapítványa ötmillió forinttal támogatja, melyet az egyes táborok között a napok számával és a tervezett létszámmal arányosan osztanak el. Írjuk ezt az összeget az *N1*-es cellába, és határozzuk meg másolható képlettel az egyes táborokra jutó támogatás összegét a *K* oszlopban! Az adatokat függvény segítségével kerekítsük ezer forintra! Melyik tábor kapta a legnagyobb támogatást? A tábor kódját jelenítsük meg képlettel az *M2*-es cellában!
2. Egy áruház parkolójában érkezéskor és távozáskor a parkolórendszer rögzíti az áthaladás óráját és percét. A parkolási díj minden megkezdett óra után egységesen 450 Ft. A táblázat egy részletét az ábrán láthatjuk. Mennyi a fizetendő parkolási díj? A feladatra az *F* oszlopban adjunk több megoldást!

	A	B	C	D	E	F
1	Rendszám	Érkezés		Távozás		Fizetendő
2		Óra	Perc	Óra	Perc	
3	KTGH4554	11	12	16	23	2 700 Ft
4	LHGV1234	11	12	13	13	1 350 Ft
5	ALMA1299	11	13	14	13	1 350 Ft
6	CIPO4434	11	13	11	48	450 Ft

3. Keressünk (nem hivatalos) statisztikai adatokat a Wikipédia oldalain az Európai Unió egyes országainak népességére! Illesszük be a kapott adatokat vágólapon át a táblázatkezelő program munkalapjára! Az adatok helyes számformátumban, de minden más cellaformátumtól mentesen jelenjenek meg!
4. Sajnos az iskola weboldalára tervezett online jelentkezési lap nem készült el időben. Hozzunk létre ezért egy megosztott táblázatot, ahol a tanulók online jelentkezhetnek az egyes táborokba a tábor kódjának, nevüknek és osztályuknak megadásával! A negyedik oszlopban jelenjen meg a *BETELT* üzenet, ha az adott táborba tervezett létszámot a korábbi jelentkezők már meghaladták!
5. A táblázatkezelők nagyon sok, különleges funkciót is tartalmaznak, kitekintésként próbáljuk ki *frissíthető webes lekérdezés* létrehozását! A frissíthető webes lekérdezés az internetről származó adatokat (például tőzsdei adatok) úgy szúrja egy adattáblába, hogy azok adott időközönként automatikusan frissülnek. Készítsünk frissíthető webes lekérdezést, amely a Magyar Nemzeti Bank honlapjáról folyamatosan megjeleníti a legfontosabb valuták középárfolyamát!

Az adatok grafikus ábrázolása (ismétlés, kiegészítés)

A leggyakoribb diagramtípusok

Ebben a leckében átismételjük a diagramok készítéséről tanultakat. Az adatokat a könyv weboldaláról letöltött `diagram.xlsx` állomány tartalmazza.

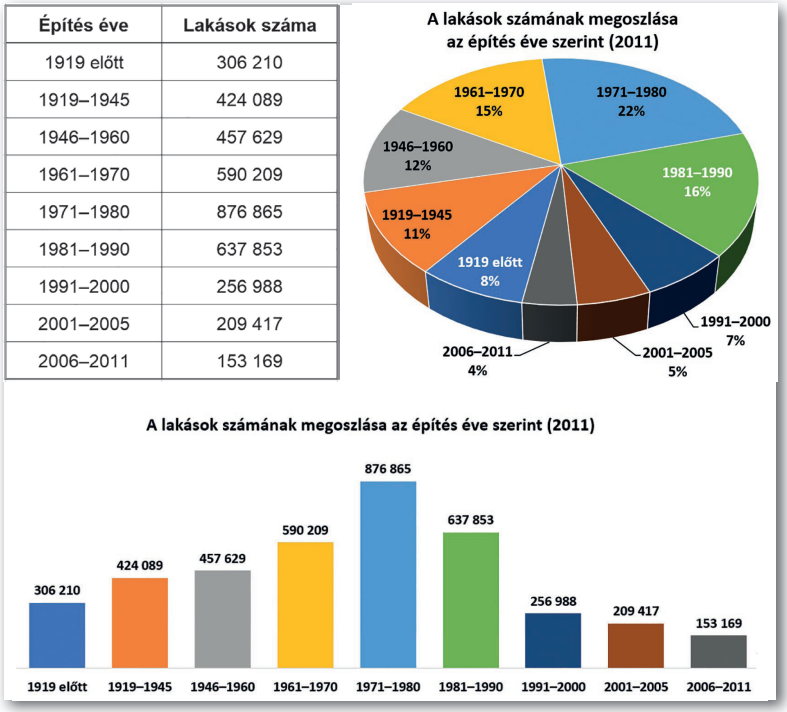
Ha az adatokat diagramon szemléltetjük, akkor az összefüggések könnyebben értelmezhetők, mint a puszta adatsorok tanulmányozásával. Leggyakrabban a következő diagramokkal találkozunk:

- a **kördiagram** csak egy adatsort mutat be, de lehetővé teszi az egyes adatok egymáshoz és az adatok összegéhez való viszonyítását is;
- az **oszlopdiagram** lehetővé teszi egy vagy több adatsor időbeli változásának követését és az adatok összehasonlítását is;
- végül a **pontdiagram** az adatsorok változását a természettudományokban megszokott módon, grafikonon szemlélteti.

5. példa: A lakásállomány megoszlása építési év szerint

Első feladatunkban a lakásállomány megoszlását tanulmányozhatjuk az építés éve szerint a 2011-es népszámlálás adatai alapján. Ábrázoljuk az adatokat önállóan kördiagramon, illetve oszlopdiagramon az alábbi mintáknak megfelelően! Vajon megtévesztő lenne-e, ha az adatokat grafikonon is szemléltetnénk?

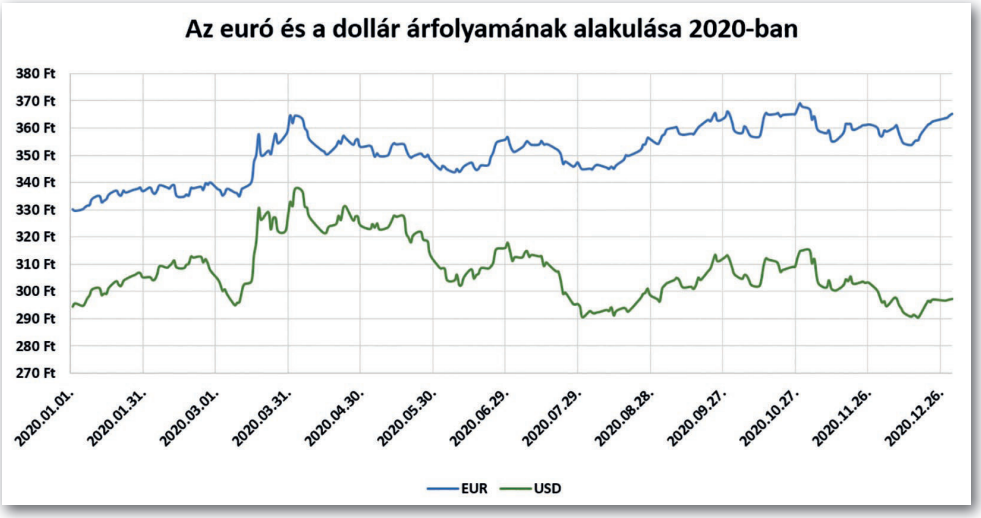
Építés éve	Lakások száma
1919 előtt	306 210
1919–1945	424 089
1946–1960	457 629
1961–1970	590 209
1971–1980	876 865
1981–1990	637 853
1991–2000	256 988
2001–2005	209 417
2006–2011	153 169



▶ A lakások számának megoszlása a 2011-es népszámlálás adatai alapján az építés éve szerint
(Forrás: http://www.ksh.hu/nepszamlalas/tablak_teruleti_00)

6. példa: Pontdiagram, grafikon

Az alábbi ábra az euró és a dollár árfolyamának napi változását szemlélteti a forinthoz képest 2020 során a Magyar Nemzeti Bank hivatalos adatai alapján. A feliratok a vízszintes tengelyen 30 napos időközzel szerepelnek, a függőleges tengelyen 270 Ft-tól 380 Ft-ig terjednek. Azzal, hogy a függőleges tengelyen az értékek nem 0-tól kezdve vannak feltüntetve, a változásokat a diagram még jobban kiemeli – de ezzel lehetővé teszi az adatok esetleges megtévesztő szemléltetését is. Készítsük el a grafikont!



► Az euró és a dollár árfolyamának alakulása 2020-ban a forinthoz képest *(Forrás: https://www.mnb.hu/arfolyam-lekerdezes)*

Feladatok

1. Készítsük el Magyarország korfáját a legfrissebb népszámlálási adatok alapján! (A korfa a népesség nemenkénti koreloszlását mutatja – részletesebben megismerkedhettünk vele földrajztanulmányaink során.) A 2011-es és a korábbi népszámlálás adatait elérhetjük például a Központi Statisztikai hivatal honlapjáról: *http://www.ksh.hu/nepszamlalas/docs/tablak/demografia/04_01_01_04.xls.*

2. A Titius–Bode-szabályt két német csillagász (Johann Daniel Titius és Johann Elert Bode) állította fel 1768-ban. Eszerint, ha a Nap és a Föld távolságát 1-nek vesszük, akkor az egyes bolygók Naptól mért távolságát az $r = 0{,}4 + 0{,}3 \cdot 2^n$ képlet adja. (A Merkúr esetén a $-\infty$ egy nagy abszolút értékű negatív számot jelent, például –1000.) Határozzuk meg képlet segítségével a *C* oszlop adatait! Szemléltessük pontdiagramon az elméleti értékeket görbített vonallal, míg a mért értékeket pontokkal, 0-tól kezdve az *n* függvényében! *(Forrás: https://www.konkoly.hu/staff/holl/bolygok/bolygok.pdf)*

	A	B	C	D
1		n	T-B szerint	mérés szerint
2	Merkúr	(-∝)	0,4	0,387
3	Vénusz	0	0,7	0,723
4	Föld	1	1	1
5	Mars	2	1,6	1,524
6	(Ceres)	3	2,8	2,77
7	Jupiter	4	5,2	5,203
8	Szaturnusz	5	10	9,537
9	Uránusz	6	19,6	19,191
10	Neptunusz	7	38,8	30,06
11	(Plútó)	8	77,2	39,44

Lépcsőfutás

7. példa: Az iskolai lépcsőfutó verseny

Toronyházakban sokfelé rendeznek lépcsőfutó versenyt. Hazánkban az egyik legismertebb lépcsőfutó bajnokságot a tűzoltóság tagjainak szervezik a Semmelweis Orvostudományi Egyetem Nagyvárad téri épületében. A versenyzők teljes védőfelszereléssel és légzőkészülékkel mintegy 510 lépcsőfokon át futnak fel a 22. emeletre.

A verseny mintájára az Irka Iskola Diákönkormányzata is megszervezi az iskolai lépcsőfutó versenyt; a tanulóknak a hivatalos tankönyveiket tartalmazó hátizsákkal kell megtenniük a távot. A verseny győztese nem az, aki a leghamarabb ér fel, hanem akinek – a fizika törvényei szerint számítva – a legnagyobb a teljesítménye.

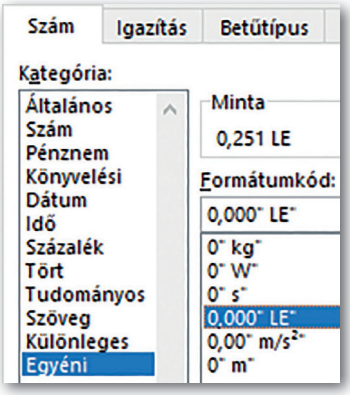
Az épületben a harmadik emeletre 114 lépcsőfok vezet, s mivel egy lépcsőfok 13 cm, ez h = 14,82 m szintkülönbséget jelent. Fizikatanulmányainkból tudjuk, hogy ha egy m tömegű testet (esetünkben egy tanulót a tankönyveivel együtt) állandó sebességgel h magasságra emelünk, akkor $W = mgh$ emelési munkát végzünk. Ha ez t idő alatt történik, akkor teljesítményünk $P = mgh/t$. A teljesítmény egysége a fizikában a watt, de találkozhatunk a lóerővel (LE) is, amely nagyjából egy ló teljesítményének felel meg: 1 LE = 746 W. Az iskolában a verseny eredményét mindig LE-ben közlik.

Az idei tanévben összesen 30 tanuló indult a versenyen; az adatokat a tankönyv weboldaláról letöltött `lepcsofutas.xlsx` állományban találjuk. A névsorban első tanuló adatai a 3. sorban, az utolsóé pedig a 32. sorban találhatók. A *B* oszlopban a tanuló tömege szerepel (a tankönyveivel együtt), a *C* oszlopban pedig a felfutás ideje. Végül a két konstanst (g és h értékét) a *D1*-es és *F1*-es cella tartalmazza.

A fentiek alapján a $P = mgh/t$ összefüggésnek megfelelően az *E3*-as cellában a másolható `=C3*$D$1*$F$1/D3` képlet szerepel, míg az *F3*-es cellában az `=E3/746`.

	A	B	C	D	E	F
1	**Lépcsőfutás**			g= 9,81 m/s²	h= 15 m	
2	**név**	**nem**	**m**	**t**	**P [W]**	**P [LE]**
3	Adrián	F	72 kg	56 s	187 W	0,251 LE
4	Albert	F	72 kg	43 s	243 W	0,326 LE
5	Alvin	F	76 kg	44 s	251 W	0,337 LE
6	Délia	L	52 kg	48 s	157 W	0,211 LE
7	Dominik	F	74 kg	51 s	211 W	0,283 LE
8	Döme	F	72 kg	53 s	198 W	0,265 LE
9	Emília	L	55 kg	46 s	174 W	0,233 LE

▶ A lépcsőfutó verseny eredménye



▶ Az egyéni számformátumok a példában (Microsoft Excel)

Egyéni számformátum kialakítása

Bár a táblázatkezelő programok nagyon sok beépített számformátumot ismernek, mindenre nem lehetnek felkészítve. Ilyen esetekben lehetőségünk van egyéni számformátumot létrehozni: a példánkban minden fizikai mennyiséget egyénileg definiáltunk. Az egyéni számformátumot formátumkóddal adhatjuk meg a *Kezdőlap > Szám > További számformátumok > Egyéni* (illetve a *Formátum > Cellák > Számok*) ablakban.

A formátumkódban a számjegyek megjelenítését a "0" (a vezető nullák is jelenjenek meg) és a "#" (a vezető nullák ne jelenjenek meg) karakterekkel, az ezres tagolást pedig a szóköz beszúrásával írhatjuk elő.

A mértékegységet a formátumkód végére idézőjelek közé kell beillesztenünk (ügyeljünk az elválasztó szóközre!). De további lehetőségeink is vannak: a már megismert dátumformátumkódokhoz hasonlóan formázhatjuk az időt, illetve a megfelelő karakterek beszúrásával akár „telefonszám-formátumot” is ki tudunk alakítani.

a cella tartalma	formátumkód	megjelenő adat
1,2345	00,00	01,23
1,2345	#0,00	1,23
12345678	### ##0	12 345 678

a cella tartalma	formátumkód	megjelenő adat
0,12	0,000" LE"	0,120 LE
2022.03.15	nnnn	kedd
3630123456	+00-00-000-000	+36-30-123-456

► Példák egyéni számformátumok kialakítására

8. példa: A verseny eredményei

A verseny eredményét külön hirdetik meg a fiúk és a lányok esetében. Az eredményeket az ismert „feltételes” statisztikai függvényekkel adhatjuk meg.

	H	I	J	K	L	M	N	O
1							J3:	=DARABHATÖBB($B$3:$B$32;I3)
2		Kód	Résztvevő	Átlagidő	Legjobb telj.	Győztes	K3:	=ÁTLAGHATÖBB($D$3:$D$32;$B$3:$B$32;I3)
3	Fiúk	F	15 fő	53,07 s	0,341 LE	Tibor	L3:	=MAXHA($F$3:$F$32;$B$3:$B$32;I3)
4	Lányok	L	15 fő	55,47 s	0,306 LE	Lia	M3:	=XKERES(L3;$F$3:$F$32;$A$3:$A$32;;0)
5							vagy	=INDEX($A$3:$A$32; HOL.VAN(L3;$F$3:$F$32;0))

► A lépcsőfutó verseny eredményei. A megfelelő képleteket az N1:O4 tartomány tartalmazza

A feltételes statisztikai függvények (**SZUMHATÖBB, ÁTLAGHATÖBB, MAXHA, MINHA** stb.) első paramétere az a tartomány, amelyen az adott statisztikai műveletet végezzük, ezt követi az a tartomány, amelyre a feltétel vonatkozik (kritériumtartomány), majd maga a feltétel. Feltételekből több is lehet egymás után. Értelemszerűen a **DARABHATÖBB** függvény esetében csak a feltételek szerepelnek.

Példánkban az egyes tartományokat abszolút cellahivatkozással adtuk meg, hogy azt másolni lehessen. Így például a *K3*-as cellában szereplő képlet:

```
=ÁTLAGHATÖBB($D$3:$D$32;$B$3:$B$32;I3)
```

Megjegyzés: A „HATÖBB” függvényekhez hasonló problémát oldanak meg az adatbáziskezelő („AB”) függvények is, amelyekkel a fejezet végén találkozunk.

A legjobb teljesítményt elérő lány vagy fiú nevét keresési függvényekkel adhatjuk meg. Ilyenek például: **FKERES, INDEX** és a **HOL.VAN** stb. Sajnos a keresési függvények a különböző táblázatkezelő programokban néha eltérő módon használandók. Az alábbi példákban feltételezzük, hogy a maximális teljesítmények csak egyszer szerepelnek.

Az **INDEX** és a **HOL.VAN** függvénypárosban a `HOL.VAN(L3;F3:F32;0)` megkeresi, hogy az *F3:F32* tartomány hányadik cellájában szerepel az *L3* cella tartalma, míg a 0 paraméter a pontos egyezést írja elő. Az `INDEX(A3:A32;…` pedig visszaadja az *A3:A32* tartomány annyiadik cellájában szereplő nevet (az **INDEX** függvény harmadik paramétere elhagyható):

```
=INDEX($A$3:$A$32; HOL.VAN(L3;$F$3:$F$32;0);1)
```

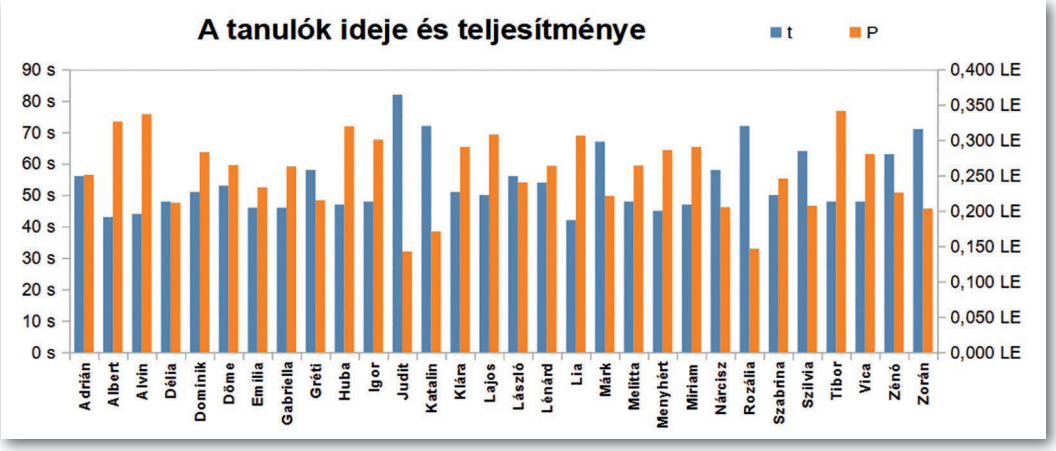
Hasonló módon működik az **XKERES** függvény is az újabb táblázatkezelő programokban. Megkeresi, hogy az *L3*-as cella tartalma az *F3:F32* tartomány hányadik cellájában van, és visszaadja az *A3:A32* tartományból az annyiadik nevet. Az 5. paraméterként szereplő 0 itt is a pontos keresésre utal.

```
=XKERES(L3;F3:F32;A3:A32;;0)
```

Érdemes megjegyeznünk, hogy mindkét esetben eltérhet a két tartomány „alakja”, például az egyik lehet egy sor, a másik egy oszlop része is.

Feladatok

1. Szemléltessük a lépcsőfutó versenyben részt vevő tanulók versenyidejét és teljesítményét közös diagramon! Mivel két különböző fizikai mennyiségről van szó, az egyik értéket (például a teljesítményt) rendeljük a második tengelyhez! Ezt a Microsoft Excelben a *Diagramtervezés* > *Más diagramtípus* > *Kombináltak,* a Calcban a *Beszúrás* > *Tengelyek* opcióval tehetjük meg. (Az ábrán látható minta LibreOffice Calcban készült.)

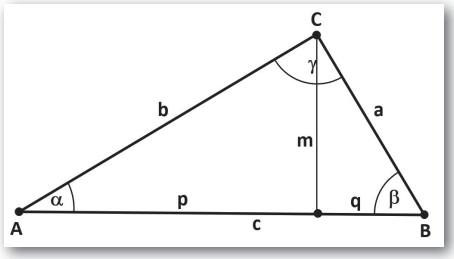


2. A fiatalabb diákok kérésére a diákönkormányzat a versenyt a következő évben már életkor szerinti kategóriákban hirdeti meg.
   a) Készítsünk ehhez egy mintát úgy, hogy a táblázatot kiegészítjük az *életkor* oszloppal is (az adatok véletlenszerűen legyenek 15, 16, 17 vagy 18)!
   b) Határozzuk meg ebben az esetben a legjobb eredményt életkoronként, azon belül pedig nemenként! Hogyan módosulnak a példában szereplő képletek?

9. példa: Derékszögű háromszögek vizsgálata

A geometriában gyakori feladat, hogy egy háromszög néhány adatának ismeretében további adatait kell kiszámítanunk. Példánkban egy derékszögű háromszöget ($\gamma = 90°$) fogunk vizsgálni.

Határozzuk meg az alábbi táblázat minden sorában a szürke hátterű cellákban szereplő adatokat az adott sorban megadott értékek alapján! A táblázatot a tankönyv weblapjáról letöltött `haromszog.xlsx` fájl tartalmazza.



	A	B	C	D	E	F	G	H
1	**a**	**b**	**c**	α (fok)	β (fok)	**m**	**p**	**q**
2	5	12				13		
3			3	30				
4		6		30				
5	5	12						
6	5		15					
7			8				5	
8	8			36				
9		24	25					

► A szürke hátterű cellák értékét kell meghatároznunk az adott sorban szereplő értékekből

Az első sorban a háromszög két befogója adott, és ki kell számolnunk az átfogóját. Ehhez Püthagorasz tételét használjuk: $a^2 + b^2 = c^2$, amiből a *C2*-es cellába az alábbi képlet adódik:

```
=GYÖK(A2*A2+B2*B2)
```

A képletben szereplő **GYÖK** függvény egy nemnegatív szám négyzetgyökét adja vissza. Hatványozásra a szám önmagával vett szorzata helyett a **HATVÁNY(a; k)** függvényt is használhattuk volna, ahol az *a* a hatványalap, a *k* pedig a kitevő:

```
=GYÖK(HATVÁNY(A2;2) + HATVÁNY(B2;2))
```

Megjegyzés: A **HATVÁNY** függvény nemcsak egész kitevő esetén használható, például az `=HATVÁNY(16;-0,5)` képlet értéke 0,25, mivel $16^{-0,5} = 0,25$. Ugyanígy, egy szám logaritmusának kiszámítására a **LOG(é; a)** függvényt használhatjuk, ahol *é* a vizsgált szám (hatványérték), *a* pedig az alap. A fenti példát alkalmazva az `=LOG(0,25; 16)` képlet eredménye –0,5. A táblázatkezelők ismerik a tízes alapú logaritmusfüggvényt is: `LOG10`. Például az `=LOG10(1000)` képlet eredménye 3, vagy az `=LOG10(0,001)` képleté –3.

A 3–4. sorban szögfüggvényekre van szükségünk. Mint ismeretes, az ábra jelöléseivel: $\sin \alpha = a/c$, $\cos \alpha = b/c$, illetve $\text{tg}\, \alpha = a/b$. A magyar nyelvű táblázatkezelő programokban a megfelelő szögfüggvények rendre **SIN**, **COS**, **TAN**. Sajnos a képletünk egy kicsit bonyolultabb lesz, mivel a szögeket a feladatban fokokban adták meg, ezzel szemben a fenti szögfüggvények azokat radiánban várják. Ezért a fokokban megadott értéket előbb a **RADIÁN** függvénnyel át kell váltanunk. A megfelelő cellák tartalma:

`A3: =C3*SIN(RADIÁN(D3))` `B3: =C3*COS(RADIÁN(D3))`
`A4: =B4*TAN(RADIÁN(D4))`

Az 5. sorban a két befogó ismeretében a szöget kell meghatároznunk, vagyis a tangensfüggvényt „visszafelé” kell alkalmaznunk: a szöget kell megadnunk a szögfüggvény (tg $\alpha = a/b$) értékéből. A megfelelő függvény az **ARCTAN**.

Az ARCTAN függvény azonban a szög értékét radiánban adja, amit így a **FOK** függvénnyel vissza kell váltani fokokba, tehát az *A4*-es cellába kerülő képlet:

`=FOK(ARCTAN(A5/B5))`

Hasonló módon kell eljárnunk a 6. sorban a szinusz-, illetve a koszinuszfüggvények „visszafelé” történő alkalmazásával. Esetünkben az **ARCSIN** és az **ARCCOS** függvények adják meg a megfelelő hegyesszögeket:

`D6: =FOK(ARCSIN(A6/C6))` `E5: =FOK(ARCCOS(A6/C6))`

Végül a 7. sorban egy összetett problémát kell megoldanunk, ezúttal ugyanis a háromszög átfogója és az egyik befogó átfogóra eső merőleges vetületét ismerjük.

Nyilván a másik átfogó merőleges vetülete a *H7*-es cellában: `=C7-G7`.

Az *F7*-es cella tartalmát a magasságtételből határozhatjuk meg (a derékszögű háromszög magassága a befogók átfogóra eső merőleges vetületeinek mértani közepe), ami a már ismert függvények alkalmazásával:

`=GYÖK(G7*H7)`

Végül az *A7*-es és a *B7*-es cellák értékét többféleképpen, például a befogótételből is megkaphatjuk (a befogó mértani közepe az átfogónak és a befogó átfogóra eső merőleges vetületének):

`A7: = GYÖK(C7*H7)` `B7: = GYÖK(C7*G7)`

Feladatok

1. A táblázat 8. sorában adott a derékszögű háromszög egyik befogója és a vele szemközti hegyesszög. Határozzuk meg a másik befogót, az átfogót, a befogók átfogóra eső merőleges vetületeit és a magasságot!
2. A táblázat 9. sorában képlet segítségével határozzuk meg a derékszögű háromszög egyik befogójából és az átfogójából a másik befogót és a hegyesszögeit!
3. Ábrázoljuk a szinusz- és a koszinuszfüggvényeket pontdiagramon, közös koordináta-rendszerben! A vízszintes tengelyen a szögek értéke fokokban jelenjen meg!
4. A tizedik évfolyamon megismerkedtünk a célértékkereséssel. Oldjuk meg célértékkeresés segítségével a következő egyenleteket:
   a) $x + \sin x = 1$ b) $x + \lg x = 10$ c) $x + x_3 = 100$

Pénzügyi számítások

10. példa: Kamatos kamat

Egy bank ötéves, sávos kamatozású betétet hirdet: a betett összeg után az első évben 4% kamatot fizet, majd onnan kezdve minden évben 0,5%-kal többet, mint az előző évben. A kamatot minden év végén hozzáírják a betéthez, így azt követően már a kamat is kamatozik. Mennyi pénzünk lesz öt év múlva, ha 100 000 Ft-ot kötöttünk le?

Hozzuk létre az ábrán látható táblázatot! Érdemes az adatok bevitelét a kitöltés műveletével gyorsítanunk (például beírunk a *C2*-es cellába 4%-ot, a *C3*-asba 4,5%-ot, majd a két cella kijelölése után a jobb alsó sarokban lévő négyzetet húzzuk).

	A	B	C	D	E
1		**tőke (év elején)**	**kamatláb**	**kamat**	**tőke (év végén)**
2	1. év	100 000 Ft	4,00%	4 000 Ft	104 000 Ft
3	2. év	104 000 Ft	4,50%	4 680 Ft	108 680 Ft
4	3. év	108 680 Ft	5,00%	5 434 Ft	114 114 Ft
5	4. év	114 114 Ft	5,50%	6 276 Ft	120 390 Ft
6	5. év	120 390 Ft	6,00%	7 223 Ft	127 614 Ft

► Kamatos kamat számítása

Az első évben a pénzünk a megadott kamatlábbal nőtt, így a *D2*-es cellába kerülő éves kamat `=B2*C2`. Év végén ezt hozzáírják a tőkéhez, így az *E2*-es cellába az `=B2+D2`, a *B3*-as cellába pedig (a következő év elején) az `=E2` képlet kerül. A képletek másolásával évente követhetjük pénzünk gyarapodását: az ötödik év végén 127 614 Ft-unk lesz.

11. példa: Lakáshitel törlesztése

Vajon mennyi törlesztőrészletet kell fizetnünk, ha lakásvásárláshoz 10 000 000 Ft hitelt vettünk fel 20 évre, fix 4%-os éves kamattal? A megoldást az ábrán láthatjuk. A feladat megoldásához a **RÉSZLET** függvényt használtuk.

	A	B
1	Kölcsön (Ft)	10 000 000
2	Éves kamat	4%
3	Futamidő (év)	20
4	**Éves részlet (Ft)**	**-735 818**
5	Teljes visszafizetés (Ft)	-14 716 350
6	**Havi részlet (Ft)**	**-60 598**
7	Teljes visszafizetés (Ft)	-14 543 528

► Lakáshitel törlesztőrészletének meghatározása

A **RÉSZLET** függvény első paramétere a kamatláb (évi 4%), második a futamidő (20 év), harmadik a jelenlegi érték (10 millió Ft). A *B4*-es cellába kerülő képlet tehát `=RÉSZLET(B2;B3;B1)`, amelynek értéke –735 818 Ft (negatív, hiszen kifizetjük). Ebben az esetben azonban azt feltételeztük, hogy a bank évente csak egyszer írja jóvá a befizetett összeget, így mi is csak évente egyszer törlesztünk.

A valóságban azonban az ügyfelek havonta törlesztenek, és azt a bank havonta írja jóvá, így arra már az év során nem számol fel kamatot. Ekkor a havi kamatlábra lesz szükségünk (amely lineáris kamatlábat feltételezve 4%/12), és a futamidőt is hónapokban kell megadnunk. A *B6*-os cellába tehát az `=RÉSZLET(B2/12;B3*12;B1)` képlet kerül, amelynek értéke –60 598 Ft.

A **RÉSZLET** függvény megadja, hogy mennyi a *törlesztőrészlet* fix *kamatláb* esetén, ha a *futamidő* végén a *jelenlegi értékből* a *jövőbeni érték* lesz:

RÉSZLET*(kamatláb; futamidő; jelenlegi érték; jövőbeni érték; típus)*

Ha a jövőbeni érték 0, akkor az elhagyható. Ha a törlesztés a fizetési periódusok elején esedékes, akkor a *típus* értéke 1, ellenkező esetben elhagyható.

12. példa: Előtakarékosság

Előbb-utóbb mindenkinek gondolnia kell a nyugdíjas éveire (még ha az olvasótól ez most még igen messze áll is), és érdemes előre takarékoskodnia. Ebben a példában erre nézünk meg egy esetet.

A nyugdíjba vonulásig hátralévő 15 évünkben havi 30 000 Ft-ot tudunk félretenni. Menynyi pénzünk lesz, amikor elérjük a nyugdíjkorhatárt, ha pénzünket olyan befektetésbe helyezzük, amely 7%-os éves hozamot fizet? Ezúttal a **JBÉ** függvényt használjuk, amely megadja egy befektetés *jövőbeni értékét* az adott *futamidő* végén, fix *kamatláb* és fix *részlet* mellett (ha a jelenlegi érték 0, akkor az elhagyható):

JBÉ*(kamatláb; futamidő; részlet; jelenlegi érték; típus)*

Ezúttal is feltételezzük, hogy a befizetett összeget a bank havonta írja jóvá, és az már az év további részében is kamatozik. Így most a *B5*-ös cellába a következő képlet került: `=JBÉ(B1/12;B2*12;B3)`. Az ábráról leolvasható, hogy nyugdíjba vonulásunkkor 9 508 869 Ft-unk lesz, miközben összesen csak 5 400 000 Ft-ot fizettünk be.

	A	B
1	Éves kamat	7%
2	Futamidő (év)	15
3	Havi befizetés (Ft)	-30 000
4	**Jövőbeni érték (Ft)**	**9 508 869**
5	Összes befizetés (Ft)	-5 400 000

► Pénzt gyűjtünk nyugdíjas éveinkre

13. példa: Áruhitel kamatlába

Egy nagy képernyős televízió ára 150 000 Ft. Az eladó felajánlja, hogy a tévét elvihetjük önrész nélkül, 3 éves, havi 6000 Ft-os törlesztőrészlettel. Vajon ez hány százalékos éves kamatot jelent? A kamatlábat a **RÁTA** függvénnyel határozhatjuk meg:

RÁTA*(futamidő; részlet; jelenlegi érték; jövőbeni érték; típus)*

Ezúttal is feltételezzük, hogy a kereskedő a törlesztést havonta írja jóvá, így a *B4*-es cellába kerülő, éves kamatot megadó képlet: `=RÁTA(B2*12;B3;B1)*12`.

	A	B
1	Kölcsön (Ft)	150 000
2	Futamidő (év)	3
3	Havi befizetés (Ft)	-6 000
4	Kamatláb (éves)	25,45%
5	Teljes befizetés (Ft)	-216 000

► Áruhitel

Feladatok

1. Egy külföldi körútra gyűjtünk. Már van 250 000 Ft-unk, de 3 év múlva összesen 1 200 000 Ft-ra lesz szükségünk. Mennyit kell ehhez havonta félretennünk, ha a bank évi 2%-os kamatot fizet?
2. Ha az előtakarékosság során összegyűjtött pénzünket havi 30 000 Ft-os részletekben szeretnénk felvenni, mennyi ideig lesz az elegendő? Tegyük fel, hogy a bankban maradó pénzünk továbbra is éves 7% kamatot hoz.
3. Készítsünk táblázatot, amely megadja, hogy 10 000 000 Ft hitel esetén mennyi a havi törlesztőrészlet! A táblázat soraiba az évek kerüljenek (5 év, 10 év… 30 év), oszlopaiba pedig a kamatlábak (2%, 4%... 12%). Milyen következtetéseket vonhatunk le a táblázat adatainak elemzéséből?

Nagy adathalmazok kezelése

Nagy adathalmazok esetén többnyire nem a megfelelő képlet kialakítása vagy az összefüggések diagramon történő ábrázolása okozza az elsődleges problémát, hanem már a szükséges adatok megkeresése is. Erre – egyszerűbb szerkezetű adathalmazok esetén – a táblázatkezelő programok is tartalmaznak eszközöket. Nagy adathalmazok feldolgozásával az *Adatbázis-kezelés* fejezetben találkozunk.

14. példa: A felvételi adatok adathalmaza

Példánkban az Irka Gimnázium felvételi adatait elemezzük. Az iskolába felvételi vizsgával juthatnak be a tanulók, a vizsga során dolgozatot írnak matematikából és magyarból, valamint szóbeli elbeszélgetésen vesznek részt. Az adatok összesítését a tankönyv weblapjáról letöltött `felveteli.xlsx` állomány tartalmazza. A táblázatban szereplő adatok jelentése a következő:

Kód: ötjegyű szám, amely az eljárás során a tanulókat azonosítja.
Név: a tanuló neve, azonos nevű tanulók előfordulhatnak.
Nem: a tanuló neme: *F* (fiú) vagy *L* (lány).
Tagozat: az iskola három tagozata közül az, amelyikre a tanuló jelentkezett: *általános, humán* vagy *reál*. Minden tanuló csak egy tagozatra jelentkezhet.
Nyelv1, Nyelv2: első idegen nyelvként mindenki az általános iskolában tanultat viszi tovább, második idegen nyelvként viszont megadhatja ezt a kettőt, ezek egyikét fogja tanulni. Van néhány tanuló, aki csak egy második idegen nyelvet választott.
Matematika, Magyar: a matematika, illetve a magyar írásbeli vizsgán elért pontszám, amely tantárgyanként legfeljebb 50 lehet.
Szóbeli: a szóbeli beszélgetésen kapott pontszám, amely legfeljebb 25 lehet.
Összesen: a három vizsgarészen elért pontszám összege, melyet a *J2*-es cellában az `=G2+H2+I2` másolható képlet ad meg.

	A	B	C	D	E	F	G	H	I	J
1	**Kód**	**Név**	**Nem**	**Tagozat**	**Nyelv1**	**Nyelv2**	**Matematika**	**Magyar**	**Szóbeli**	**Összesen**
2	11404	Ablonczi Zoltán	F	általános	olasz	francia	39	34	21	94
3	50080	Aigner Győző	F	reál	kínai	német	44	34	23	101
4	94944	Almási Paulina	L	általános	német	koreai	46	37	17	100
5	38216	Ambrózy Erika	L	általános	spanyol		42	46	23	111
6	54293	Ambrus Gréti	L	humán	német	német	39	48	25	112
7	61810	Babarczy Jenő	F	reál	orosz	olasz	41	37	13	91
8	85086	Balajthy Barnabás	F	általános	spanyol	latin	48	48	11	107
9	63166	Bálinth Beáta	L	reál	spanyol	német	40	37	19	96
10	51551	Ballogh Jadviga	L	reál	kínai	francia	45	44	24	113

► A felvételi vizsga adatait tartalmazó táblázat (a tanulók neve szerinti sorrendben)

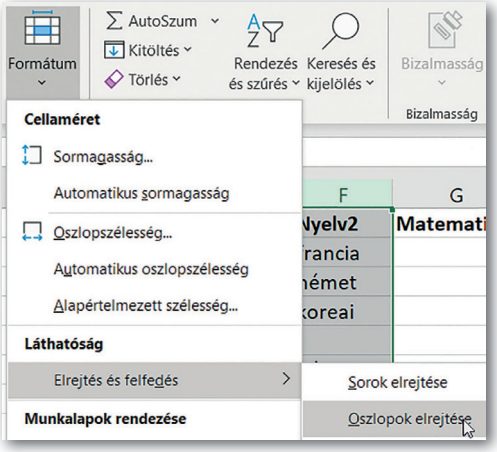
A táblázat adatsorait *rekordoknak,* oszlopait pedig *mezőknek* is nevezik. A mezőket az első sorban szereplő nevük *(mezőnév)* azonosítja. A *Kód* minden tanuló esetén eltér, így egyértelműen meghatározza a tanulót: ez a mező a *kulcs*. Az *Összesen* mezőt a többi adatból határoztuk meg, az ilyen mezőt *számított mezőnek* is nevezik.

Sorok és oszlopok

A mintaként használt adathalmaz csupán 10 mezőt és 150 rekordot tartalmaz, de egy képernyőn a teljes adathalmaz már nem jeleníthető meg olvashatóan. A működő köznevelési intézmények listája 2021-ben 48 mezőt és 5727 rekordot tartalmazott – értelemszerűen ennek áttekintése még nehezebb lehet.

Sok esetben segíti az adatok megjelenítését a képernyőn az oszlopok szélességének és a sorok magasságának megadása. Az oszlopok szélességét például a *Kezdőlap* > *Formátum* > *Oszlopszélesség* (illetve *Formátum* > *Oszlopok* > *Szélesség*) menüponttal adhatjuk meg. Kényelmesebb megoldás azonban, ha a fejlécoszlop elválasztó vonalát húzzuk az egérrel, ilyenkor zárójelben megjelenik az oszlopszélesség értéke is. Bár a táblázatkezelők az oszlopok szélességére, illetve a sorok magasságára többféle egységet is használnak, azt többnyire megadhatjuk centiméterben is. Ehhez a Microsoft Excelben például a *Nézet* menüben át kell váltanunk *Lapelrendezés* nézetre, míg a LibreOffice Calcban a centiméter az alapértelmezett.

A megjelenítést áttekinthetőbbé teszi, ha azok az oszlopok (vagy sorok), amelyekre éppen nem vagyunk kíváncsiak, nem jelennek meg. Ha például csak a tanulók eredménye érdekel bennünket, akkor esetünkben a *C:F* oszlopokat el tudjuk rejteni. Ehhez az oszlopok kijelölése után válasszuk a *Kezdőlap* > *Formátum* > *Elrejtés és felfedés* > *Oszlopok elrejtése* lehetőséget (illetve a *Formátum* > *Oszlopok* > *Elrejtés* menüpontot)! Később hasonló módon jeleníthetjük meg az elrejtett oszlopokat újra.

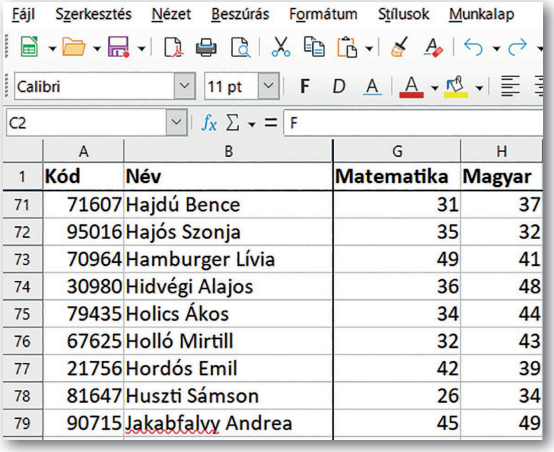


▶ Oszlopok elrejtése (Microsoft Excel)

Az ablaktábla kezelése

Ha példánkban a táblázatot felfelé görgetjük, akkor a mezőnevek egy idő után már nem látszanak, így gondot okozhat az adatok értelmezése. Ha ezt el akarjuk kerülni, akkor rögzítsük az első sort a képernyőn például a *Nézet* > *Panelek rögzítése* > *Felső sor rögzítése* (illetve a *Nézet* > *Cellák rögzítése* > *Első sor rögzítése*) menüponttal! Hasonlóan rögzíthetjük az első oszlopot is.

A táblázatkezelő programok lehetővé teszik több sor vagy több oszlop egyidejű rögzítését is. A jobb oldali ábrán például folyamatosan megjelenik az első sor és az első két oszlop. Ehhez kijelöltük az első „mozgatható” cellát, a *C2*-es cellát, majd a *Nézet* > *Panelek rögzítése* > *Ablaktábla rögzítése* (illetve a *Nézet* > *Rögzítés*) lehetőséget választottuk. A funkció ugyanezen az úton kapcsolható ki.



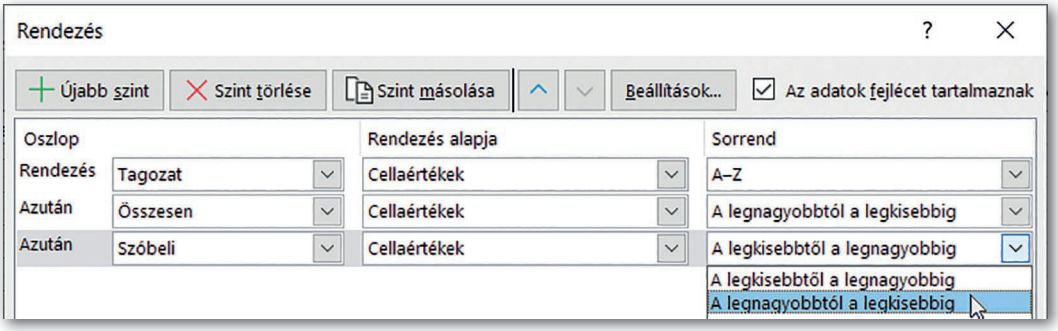
	A	B	G	H
1	Kód	Név	Matematika	Magyar
71	71607	Hajdú Bence	31	37
72	95016	Hajós Szonja	35	32
73	70964	Hamburger Lívia	49	41
74	30980	Hidvégi Alajos	36	48
75	79435	Holics Ákos	34	44
76	67625	Holló Mirtill	32	43
77	21756	Hordós Emil	42	39
78	81647	Huszti Sámson	26	34
79	90715	Jakabfalvy Andrea	45	49

▶ Görgetés adattábla rögzítésével (LibreOffice Calc)

Rendezés

A bejutás sorrendjét minden tagozaton az összpontszám alapján határozzák meg: ehhez a tanulók adatait rendezik tagozat, azon belül összpontszám szerint. Előfordulhat, hogy a tanulók összpontszáma azonos, ebben az esetben a szóbeli beszélgetés eredménye dönt. Rendezzük a tanulók adatait a fentiek alapján! (Mivel a tanulók több iskolába is jelentkezhetnek, ez még nem a felvett tanulók végleges listája!)

Rendezésnél ügyelni kell arra, hogy az összetartozó adatok mind ki legyenek jelölve, így a sorrend kialakításánál együtt mozogjanak. Szerencsére a legtöbb esetben elegendő az aktív cellát a rendezendő tartományra vinni, és a program automatikusan kijelöli a teljes tartományt. A rendezést például az *Adatok* > *Rendezés* menüpontjával indíthatjuk. A megjelenő ablakban egymás után vihetjük fel az újabb szinteket: a táblázatkezelő programok általában automatikusan felismerik és felajánlják ilyenkor a mezőneveket. Minden szintnél egyenként adhatjuk meg a rendezés irányát.

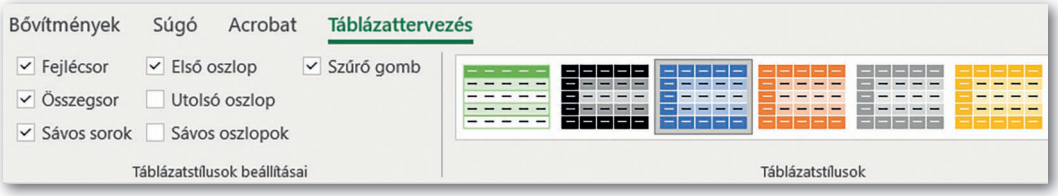


► A felvételt nyert tanulók kiválasztása rendezéssel (Microsoft Excel)

Tartomány táblázattá alakítása

A példánkban szereplő táblázat első sorában a mezőnevek vannak, első oszlopában pedig a kulcsmező adatai – így egy adatbázis-kezelő adattáblájának is megfelel a szerkezete. A Microsoft Excel különböző verziói lehetővé teszik az így elrendezett adatok gyors formázását és a gyakori statisztikai számítások gyors elvégzését.

Ahhoz, hogy ezeket a funkciókat elérjük, először az adatokat tartalmazó tartományt – az Excel szóhasználatával élve – „táblázatként kell megformáznunk", például a *Kezdőlap* > *Formázás táblázatként* menüponttal. A táblázat formátumát egy listáról választhatjuk ki. Táblázattá formázás után a mezőnevek mellett egy-egy nyílhegy jelenik meg, ezek legördítésével több funkciót, például a már megismert rendezést és szűrést is elvégezhetjük.



► Formázás táblázattá alakítás után a *Táblázattervezés* szalagon (Microsoft Excel)

Átalakítás után a táblázat szerkezetét tovább finomíthatjuk, például a *Táblázattervezés* szalagon a *Táblázatstílusok* beállításaival. Így például kiemelhetjük az első vagy utolsó oszlopot, sávos hátteret állíthatunk be, vagy bekapcsolhatjuk az *Összegsort*.

Az *Összegsor* bekapcsolása teszi lehetővé statisztikai számítások gyors elvégzését. Ha az adattábla alatt megjelenő sorban valamely cellára kattintunk, akkor egy listáról kiválaszthatjuk, hogy milyen statisztikai számításokat végezzen el a program az adott mező adataival. Például határozzuk meg így, hogy mennyi a tanulók átlagos pontszáma!

A táblázatot tartománnyá például a *Táblázattervezés* szalag *Átalakítás tartománnyá* pontjával alakíthatjuk vissza. Ilyenkor a formátumbeállítások és az összesítő függvények is megmaradnak.

	A	B	C	D	E	F	G	H	I	J
1	Kód	Név	Nem	Tagozat	Nyelv1	Nyelv2	Mat	Magy	Szóbeli	Összesen
146	11106	Wild Róza	L	reál	francia	latin	44	36	11	91
147	16882	Wotticzky Natália	L	általános	francia	latin	45	50	25	120
148	73875	Zágoni Otília	L	általános	spanyol	német	45	27	11	83
149	30683	Zucker Aurélia	L	általános	német	francia	33	28	24	85
150	52498	Zsupunszki Fedor	F	általános	kínai	spanyol	42	33	10	85
151	68981	Zsurkán Emánuel	F	humán	orosz	német	32	34	15	81
152	Összeg									94,653333
153										
154										
155										

Nincs
Átlag
Darab
Darabszám

► A tanulók átlagos pontszámának meghatározása az *Összegsor* segítségével (Microsoft Excel)

Kérdések, feladatok

1. A Microsoft Excel *Normál* nézetben szokatlan egységeket használ az oszlopok szélességére és a sorok magasságára (alapértelmezett 8,43, illetve 15). Vajon milyen egységben értendők ezek a paraméterek?
2. A `felveteli` adattáblában oldjuk meg a következő feladatokat!
   a) Mennyi a 30. tanuló összpontszáma az egyes tagozatokon?
   b) Rendezzük a felvételiző tanulók adatait a *Nyelv1,* azon belül a *Nyelv2,* végül a *Tagozat* mező szerint!
   c) Felmerült, hogy a felvételi sorrendet azonos pontszám esetén a reál tagozaton a matematika, a humán tagozaton pedig a magyar írásbeli vizsga eredménye alapján döntsék el. Hogyan lehetne ezt megvalósítani?
3. Töltsük le az Oktatási Hivatal weblapjáról a köznevelési intézmények nyilvános adatbázisát *(https://dari.oktatas.hu/kozerdeku_index)!*
   a) Hány mezőt és hány rekordot tartalmaz az adathalmaz?
   b) Rögzítsük az ablaktáblát úgy, hogy görgetéskor a mezőnevek, illetve az első két mező folyamatosan látszódjon!
   c) A megfelelő oszlopok elrejtésével állítsuk be, hogy csak az intézmények neve és címe jelenjen meg!
   d) Vizsgáljuk meg az adatokat: vajon milyen szempontrendszer alapján vannak rendezve? Módosítsuk a rendezést úgy, hogy előre kerüljenek az óvodák, majd az általános iskolák, végül a gimnáziumok! Hol jelennek meg a többcélú intézmények ebben az esetben (vagyis amelyek például óvodai és iskolai feladatokat is ellátnak)?

Adatok kiválogatása szűréssel

Adatok szűrése

Nagy adathalmazok esetén gyakran van arra szükségünk, hogy csak bizonyos feltételeknek megfelelő rekordok jelenjenek meg. A feltételt – amelyet **szűrőfeltételnek** is neveznek – megadhatjuk például egy logikai formulával így:

```
Tagozat="humán" és Magyar>=40
```

A táblázatkezelő programok általában többféle lehetőséget is tartalmaznak a szűrőfeltételek megadására. Például elegendő lehet csupán kiválasztani a megfelelő feltételeket *(automatikus szűrés)*, vagy külön be is kell írnunk azokat a munkalap celláiba *(irányított szűrés).*

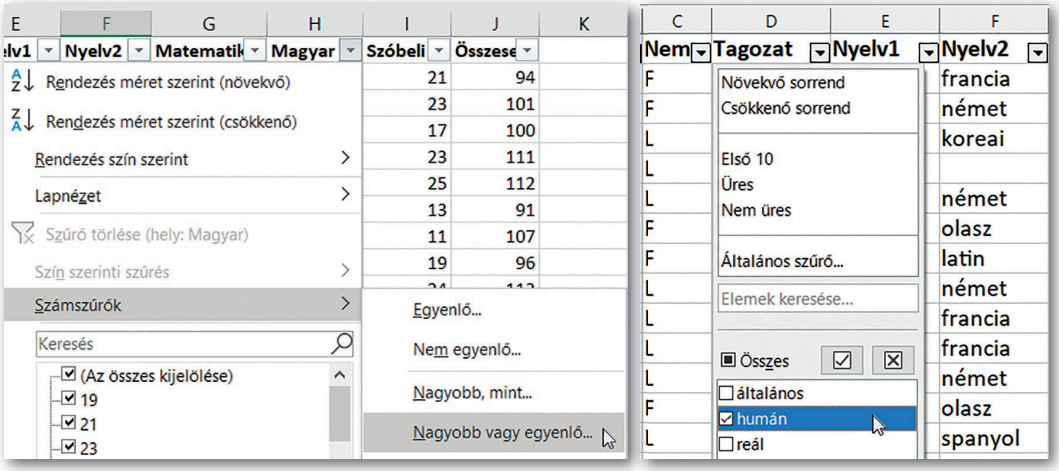
15. példa: Automatikus szűrés

Azoknak a tanulóknak, akik a humán tagozatra jelentkeztek, és a magyarírásbelin legalább 40 pontot értek el, az iskola magyartanárai egy-egy verseskötettel kedveskednek. Kik ezek a tanulók? A feladat megoldásához alkalmazzunk automatikus szűrést!

Az automatikus szűrés indításához kattintsunk az *Adatok* > *Szűrő* (illetve az *Adatok* > *Automatikus szűrő*) menüpontra! Ekkor a mezőnevek mellett megjelenik egy-egy nyílhegy, amellyel az adott mezőre vonatkozó beállításokat elvégezhetjük.

A beállítás két lépésben történik. A `Tagozat="humán"` feltétel megadásához a *Tagozat* melletti listán elegendő bejelölni a *humán* lehetőséget. A `Magyar>=40` feltétel megadásához viszont a listán válasszuk ki például a *Számszűrők* > *Nagyobb vagy egyenlő* lehetőséget (illetve az *Általános szűrő* pontot, azon belül a *Feltétel* listán a >= opciót), és a megfelelő beviteli mezőben adjuk meg a 40 pontos ponthatárt!

Hasonló módon alkalmazhatjuk szöveges adatok esetén a *Szövegszűrő,* illetve dátumot tartalmazó mező esetén a *Dátumszűrő* menüpontot.



► A tagozatot elegendő kiválasztanunk a listáról (balra: LibreOffice Calc), míg a pontszámra vonatkozó feltételt a megfelelő számszűrő alkalmazásával adhatjuk meg (jobbra: Microsoft Excel)

16. példa: Irányított szűrés

Az iskola nyári felzárkóztató foglalkozást szervez azoknak a tanulóknak, akik reál tagozatra jelentkeztek, de a matematika írásbeli eredményük nem érte el a 30 pontot, vagy humán tagozatra jelentkeztek, és a magyar írásbeli dolgozatuk rosszabb lett 30 pontosnál. Válasszuk ki az érintett tanulókat irányított szűréssel!

A feltétel megfogalmazása logikai formulával (mivel az *és* művelet magasabb prioritású, mint a *vagy* művelet, a zárójelek akár el is hagyhatók):

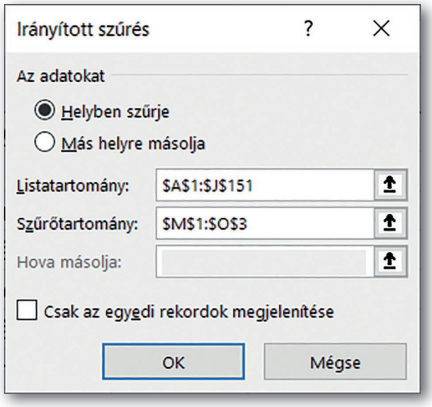
```
(Tagozat="reál" és Matematika<30) vagy
(Tagozat="humán" és Magyar<30)
```

Irányított szűrő esetén a feltételeket a *szűrőtartomány* tartalmazza. Ennek első sorában a mezőneveket kell feltüntetnünk, alatta pedig minden cella egy-egy feltételt tartalmaz az adott mezőre (üres is lehet). Fontos, hogy az egymás mellé írt feltételeknek egyszerre kell teljesülniük (tehát logikai *és* kapcsolat van közöttük), míg az egymás alá írt sorok közül elegendő az egyiknek teljesülnie (a sorok között *vagy* kapcsolat van).

	L	M	N	O
1		**Tagozat**	**Matematika**	**Magyar**
2		reál	<30	
3		humán		<30
4				

▶ A szűrőfeltétel az M1:O3 szűrőtartományban

Az irányított szűrés indításához kattintsunk az adattáblába (így a legtöbb program automatikusan felismeri az adatokat tartalmazó *listatartományt*), majd válasszuk az *Adatok* > *Irányított szűrés* (illetve az *Adatok* > *Több szűrő* > *Irányított szűrő*) lehetőséget! A megjelenő ablakban a *szűrőtartományt* megadhatjuk a tartomány beírásával, de többnyire az egér húzásával is.



▶ Az *Irányított szűrő* ablak (Microsoft Excel)

Az *Irányított szűrő* ablakban általában lehetőségünk van arra is, hogy a kigyűjtött rekordokat az adott munkalapon belül, egy megadott tartományba másolja a program.

Kérdések, feladatok

1. Vajon a pontszámok esetén miért célszerűbb számszűrő alkalmazása a megadott intervallum megadásával, mint csupán a listán egyébként is megjelenő értékek kiválasztása?
2. Azoknak a tanulóknak, akik mindkét tárgyból legalább 45 pontos dolgozatot írtak, az iskola egy külön „iskolakóstoló” foglalkozást szervez. Válasszuk ki automatikus szűréssel ezeket a tanulókat!
3. A némettanárok előzetes szintfelmérőt íratnak. Másoljuk a munkalapra az M10-es cellától kezdődően azok adatait, akik második idegen nyelvként a németet választották!
4. Nyissuk meg a köznevelési intézmények adatbázisát, és
   a) általános szűrővel keressük meg a Szegedi járás gimnáziumait!
   b) irányított szűrő segítségével a Pécsi járás óvodáit és általános iskoláit!

Adatok kiemelése feltételes formázással

Feltételes formázás

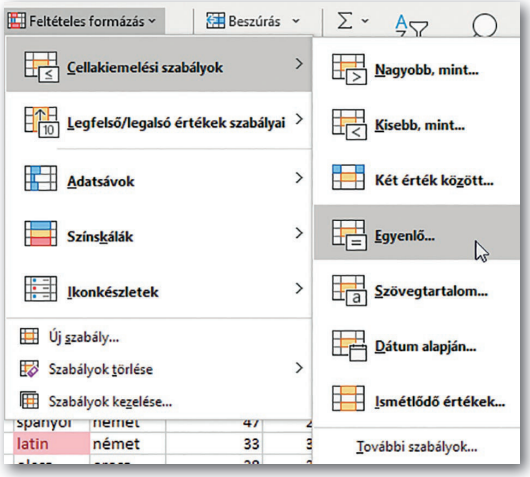
Szűrés esetén csak az adott feltételnek megfelelő adatok jelennek meg, ezzel szemben feltételes formázáskor minden cella látszik, ám az előírt feltételeknek megfelelő adatok formátuma eltérő lesz. A feltételes formázás nagy előnye, hogy az adatok módosulása esetén a formátumok azonnal frissülnek, szemben a szűrővel, ahol a frissítést kézzel kell elvégeznünk.

A feltétel megadására alapvetően kétféle lehetőségünk van: egyszerűbb esetben elegendő a program által kínált lehetőségek közül választanunk, de nagyobb szabadságot ad, ha a feltételt képlettel adjuk meg.

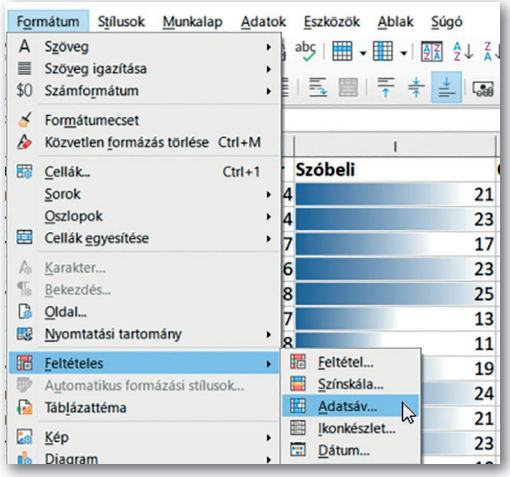
17. példa: **Feltétel megadása a feltétel kiválasztásával**

Első példánkban a latintanár kérésére megkeressük azokat a tanulókat, akik második idegen nyelvként a latint választották, és a *latin* szót pirossal kiemeljük.

A megfelelő adatok az *E:F* oszlopokban vannak, így jelöljük ki először ezt a tartományt, majd válasszuk a *Kezdőlap* > *Feltételes formázás* > *Cellakiemelési szabályok* > *Egyenlő* (illetve a *Formátum* > *Feltételes* > *Feltétel*) menüpontot! A megjelenő ablakban meg kell adnunk a cella értékét *(latin)*, és ki kell választanunk a formátumát.



► A latin nyelvet tanulók kiemelése (Microsoft Excel)



► Adatsávok alkalmazása (LibreOffice Calc)

Második példánkban az igazgató a szülői tájékoztatón szemlélteti a szóbeli beszélgetésen elért eredményeket. Az eredmények gyorsan felismerhető bemutatásához a pontszámokkal arányos szélességű csíkokat jelenít meg a megfelelő cellákban (természetesen a tanulók többi adatának elrejtésével). Vajon hogyan valósíthatjuk ezt meg?

Ezúttal az *I* oszlopot jelöljük ki, majd a *Kezdőlap* > *Feltételes formázás* > *Adatsávok* (illetve *Formátum* > *Feltételes* > *Adatsáv*) lehetőséget választjuk. A megjelenő ablakban megadhatjuk az alkalmazandó sávok színét.

A választható feltételek a táblázatkezelő programtól függően sokfélék lehetnek, de többnyire lehetőségünk van kiemelni az első valahány legnagyobb vagy legkisebb elemét; értéküktől függően eltérő színekkel vagy ikonokkal szemléltetni az adatokat, stb.

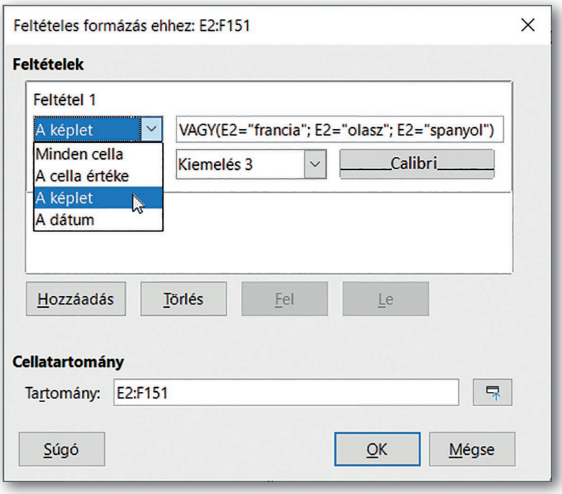
18. példa: A feltétel megadása képlettel

A nyelvtanárok azt kérik, hogy szürke háttérrel emeljük ki a választott neolatin (vagyis a francia, olasz és spanyol) nyelveket. A feltételt ezúttal képlettel adjuk meg.

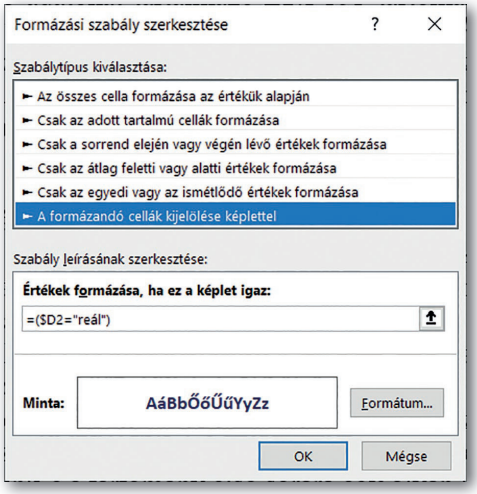
Első lépésként jelöljük ki a megfelelő adatokat tartalmazó *E2:F151* tartományt! Válasszuk a *Kezdőlap > Feltételes formázás > Új szabály > A formázandó cellák kijelölése képlettel* lehetőséget (illetve a *Formátum > Feltételes > Kezelés > Hozzáadás* gombot, majd *A cella értéke* kezdetű listán *A képlet* lehetőséget)! Végül írjuk be a megfelelő rovatba a következő képletet, és válasszuk ki az alkalmazandó formátumot!

```
=VAGY(E2="francia"; E2="olasz"; E2="spanyol")
```

Vajon mi történik a képlet végrehajtása során? A képlet relatív cellahivatkozást tartalmaz a kijelölt tartomány bal felső cellájára *(E2)*. A táblázatkezelő program ezt a képletet „végigmásolja” a kijelölt tartományra, és amely cellára a képlet IGAZ logikai értéket ad, arra alkalmazza a megadott formátumot.



▶ A neolatin nyelvek kiemelése képlettel (LibreOffice Calc)



▶ A reál tagozatosok adatainak kiemelése vegyes cellahivatkozással (Microsoft Excel)

A reál tagozat osztályfőnöke kíváncsi leendő tanítványaira. Emeljük ki a reál tagozatot választók sorait **félkövér, sötétkék** betűkkel!

Eddigi példáinkban mindig csak az adott feltételnek megfelelő adatokat formáztuk meg, ezúttal azonban a reál tagozatosok összes adatát szeretnénk kiemelni. Jelöljük tehát ki az *A2:J151* tartományt! Vajon meg tudunk-e adni a tartomány első sorára egy olyan képletet, amely egyrészt másolható (lefelé), másrészt csak a tagozatválasztás *D* oszlopára vonatkozik? Természetesen igen, hiszen erre szolgál a vegyes cellahivatkozás:

```
=($D2="reál")
```

Utolsó példánkban a humán tagozatosok leendő magyartanára szeretné megtekinteni a humán tagozatra jelentkezők magyar írásbeli eredményeit. Azt kéri, hogy ezek a pontszámok (és csak ezek!) legyenek kiemelve például sötétvörös szegéllyel.

Ezúttal a formázandó tartomány a *H2:H151,* így ezt kijelöljük, a képletet pedig a szokásos módon beírjuk:

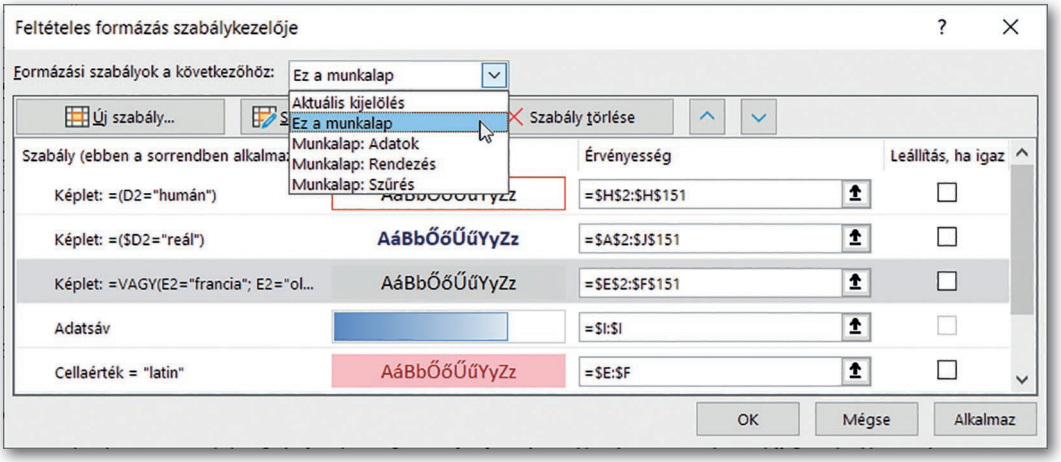
```
=(D2="humán")
```

A megvalósítás során a táblázatkezelő a képletet automatikusan végrehajtja a *D2:D151* tartományra, és megformázza a *H2:H151* tartomány megfelelő elemeit.

A szabályok kezelése

Ha egy munkalapon több feltételes formázást is alkalmazunk, előfordulhat, hogy egy adott cellára több szabály is teljesül. Például, ha a neolatin nyelvek helyett a nyelvtanárok a latin nyelvekre kíváncsiak, akkor a *latin* nyelvre két szabály is vonatkozik: a latintanár korábban már halványpirosra állíttatta a hátteret, most pedig szürkére kell. Ebben az esetben módosíthatjuk a szabályok végrehajtásának sorrendjét.

A szabályok kezelését, vagyis utólagos módosítását, sorrendjének megváltoztatását vagy akár a szabályok törlését is például a *Kezdőlap* > *Feltételes formázás* > *Szabályok kezelése* (illetve a *Formátum* > *Feltételes* > *Kezelés*) menüponttal tehetjük meg.



► A példákban szereplő feltételes formázások áttekintése (Microsoft Excel)

Feladatok

1. Képlet alkalmazása nélkül
   a) emeljük ki az összpontszámok hátterét a színskála különböző színeivel!
   b) jelöljük meg azokat, akik a legjobb, illetve a legrosszabb eredményeket érték el a matematika írásbeli dolgozat során!
2. A következő feladatokat képlet alkalmazásával oldjuk meg!
   a) Emeljük ki a fiúkat a *Nem* oszlopban sötétkék alapon fehér színű betűvel!
   b) Jelöljük meg félkövér betűvel az átlagosnál magasabb összpontszámokat!
   c) Narancsszínű, félkövér betűvel válasszuk ki a humán tagozatot választók összes adatát!
   d) Szegélyezzük azoknak a tanulóknak az azonosítóját, akik csak egy idegen nyelvet adtak meg!
3. Feltételes formázás segítségével készítsünk sakktáblát!

Részösszegképzés és kimutatás

Az adatok elemzése

Milyen eredményt értek el az írásbeli vizsgán az egyes tagozatokra jelentkező tanulók? Van-e különbség a humán és a reál tagozatosok matematika és magyar írásbeli dolgozatának pontszámai között? A fiúk vagy a lányok szerepeltek jobban a szóbeli beszélgetésen? Van-e összefüggés a nyelvválasztás és a tagozat megválasztása között?

Ezekben a kérdésekben az a közös, hogy az adatokat bizonyos szempontok szerint csoportosítva vizsgáljuk. Az ilyen problémákra a táblázatkezelő programok többféle megoldást is kínálnak, például a részösszegképzést és a kimutatást.

19. példa: Az oszlopok csoportosítása

A részösszegképzés tárgyalása előtt ismerkedjünk meg a csoportosítással! Csoportosítás során néhány egymás melletti oszlopot (vagy egymást követő sort) csoportba foglalunk: a csoportba foglalt oszlopokat egyszerre jeleníthetjük meg vagy rejthetjük el.

Például, ha a nyelveket és a felvételi vizsga részpontszámait nem szeretnénk megjeleníteni, akkor kijelöljük az *E:I* oszlopokat, majd az *Adatok* > *Tagolás* > *Csoportosítás* (illetve az *Adatok* > *Csoportosítás és vázlat* > *Csoportosítás*) pontot választjuk. A csoportot az oszlopok fölött egy vonal jelzi, a csoportot a vonal szélén lévő [–] ikonra kattintva csukhatjuk össze, illetve a [+] ikonra kattintva nyithatjuk meg. A csoportosítás többszintű is lehet, vagyis a csoporton belül is hozhatunk létre csoportokat.

	A	B	C	D	E	F	G	H	I	J
1	**Kód**	**Név**	**Nem**	**Tagozat**	**Nyelv1**	**Nyelv2**	**Matematika**	**Magyar**	**Szóbeli**	**Összesen**
2	11404	Ablonczi Zoltán	F	általános	olasz	francia	39	34	21	94
3	50080	Aigner Győző	F	reál	kínai	német	44	34	23	101

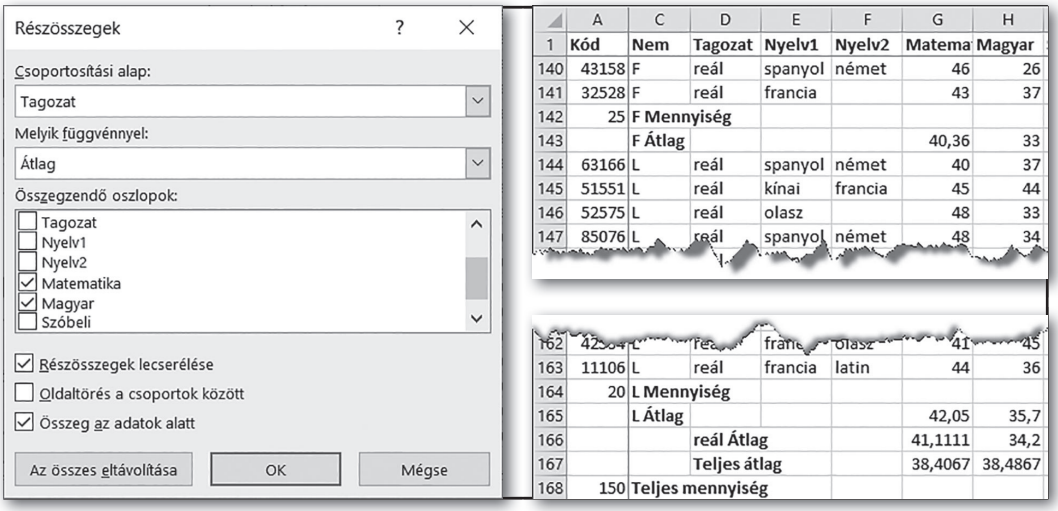
▶ Többszintű csoportosítás: az *E:I*, azon belül az *E:F* oszlopok összecsukhatók (LibreOffice Calc)

20. példa: Részösszegképzés

Milyen pontszámot értek el átlagosan a magyar- és matematikavizsgán az egyes tagozatra jelentkezők? Van-e különbség a fiúk és a lányok eredménye között?

A feladat megoldásához a tanulókat előbb tagozatonként és nemenként csoportosítani kell, majd az egyes csoportokra kell elvégezni a megfelelő számításokat. A csoportokat részösszegképzés során a táblázatkezelő programok automatikusan létrehozzák: az egymást követő, azonos értékek szerint alakítják ki őket. Ezért a csoportosítást többnyire rendezéssel elő kell készítenünk. A rendezést például az *Adatok* > *Rendezés* menüponttal végezhetjük, ezúttal a *Tagozat,* azon belül pedig a *Nem* mező szerint.

Az adatokat tartalmazó tartományt általában nem kell kijelölnünk, elegendő, ha valamely nem üres cellájára kattintunk. Az összesítést először tagozatonként végezzük. A részösszegképzés beállításához válasszuk például az *Adatok* > *Tagolás* > *Részösszeg* (illetve az *Adatok* > *Részösszeg*) menüpontot! A megjelenő ablakban meg kell adnunk a csoportosítás alapjául szolgáló mezőt *(Tagozat)*, azt a mezőt, amellyel műveletet végzünk *(Matematika, Magyar)* végül az alkalmazandó statisztikai függvényt *(Átlag)*. Az eredmény minden tagozat utolsó rekordja után, a *Magyar* és a *Matematika* oszlopában jelenik meg, a táblázat alatt pedig a teljes adathalmazra vonatkozó eredményt láthatjuk.



	A	C	D	E	F	G	H
1	Kód	Nem	Tagozat	Nyelv1	Nyelv2	Matema	Magyar
140	43158	F	reál	spanyol	német	46	26
141	32528	F	reál	francia		43	37
142	25	F Mennyiség					
143		F Átlag				40,36	33
144	63166	L	reál	spanyol	német	40	37
145	51551	L	reál	kínai	francia	45	44
146	52575	L	reál	olasz		48	33
147	85076	L	reál	spanyol	német	48	34
162	42564	L	reál	francia	olasz	41	45
163	11106	L	reál	francia	latin	44	36
164	20	L Mennyiség					
165		L Átlag				42,05	35,7
166			reál Átlag			41,1111	34,2
167			Teljes átlag			38,4067	38,4867
168	150	Teljes mennyiség					

► Részösszegképzés *Tagozat* és *Nem* szerint a *Magyar* és *Matematika* mezők átlagával (Microsoft Excel)

Ha a részösszegképzést újabb mezők vizsgálatával vagy újabb szintek bevezetésével szeretnénk bővíteni, akkor a részösszegképzést újból le kell futtatnunk. Példánkban az adatokat nemek szerinti csoportosításban tovább bővítettük, és megadtuk a tanulók számát is az *Azonosító* mező elemeinek megszámolásával.

A táblázatkezelő programok használata részben eltérő lehet. A Microsoft Excel esetén például a részösszegképzés újbóli lefuttatása esetén ügyeljünk arra, hogy az a korábbi részösszegeket ne cserélje le! A LibreOffice Calc használata esetén viszont nem szükséges az adatokat előzetesen rendezni, illetve lehetőségünk van egy lépésben többféle csoportosítást is előírni.

21. példa: Kimutatás készítése

Határozzuk meg most a felvételizők számát és a magyar-, valamint matematika-írásbeli pontszámának átlagát tagozatonként, azon belül nemenként *kimutatás* segítségével! A kimutatás az eredményeket egy táblázatban jeleníti meg, amelynek sorcímkéi a nemek, oszlopcímkéi a tagozatok, a megfelelő cellákban pedig a létszámok és az átlagpontszámok vannak.

	általános			humán			reál			Összes Fő	Összes Mat	Összes Magy
	Fő	Mat	Magy	Fő	Mat	Magy	Fő	Mat	Magy			
Fiúk	35	38	41,09	15	39	39,53	25	40	33,00	75	39	38,08
Lányok	32	38	39,44	23	34	40,91	20	42	35,70	75	38	38,89
Összesen	**67**	**38**	**40,30**	**38**	**36**	**40,37**	**45**	**41**	**34,20**	**150**	**38**	**38,49**

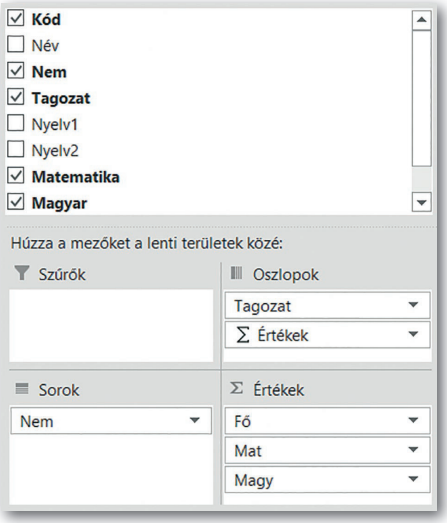
► A megformázott kimutatás (Microsoft Excel)

A kimutatás készítéséhez kattintsunk a tartomány valamely nem üres cellájára, majd válasszuk a *Beszúrás* > *Kimutatás* (illetve az *Adatok* > *Kimutatástábla* > *Beszúrás vagy szerkesztés*) menüpontot!

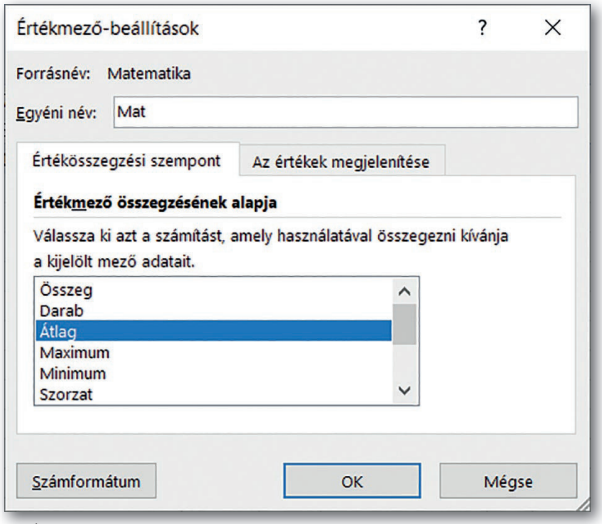
A kimutatás készítése során az egér húzásával adhatjuk meg az egyes mezők szerepét. Példánkban az *Oszlopok* (illetve *Oszlopmezők*) listára a *Tagozat* mezőt, a *Sorok* (illetve *Sormezők*) listára a *Nem* mezőt, végül az *Értékek* (illetve *Adatmezők*) listára a *Kód,* a *Matematika* és a *Magyar* mezőket húzzuk.

Az *Értékek (Adatmezők)* esetén külön meg kell adnunk a megfelelő statisztikai függvényeket. Ezt például a mező nevére egyet kattintva a megjelenő menü *Értékmező-beállítások* pontjával (illetve a mező nevére kettőt kattintva) választhatjuk ki.

A további formátumok beállítása már programonként eltérő lehet. Általában a kimutatás táblázatában a feliratokat, számformátumokat, cellaformátumokat a szokásos módon adhatjuk meg. A Microsoft Excelben azonban a kimutatásban megjelenő címkéket és a mező számformátumát az *Értékmező-beállítások* ablakban is módosíthatjuk.



▶ A mezők megadása húzással (Microsoft Excel)



▶ Értékmező-beállítások a matematika írásbeli pontszámok esetén (Microsoft Excel)

Kérdések, feladatok

1. Részösszegképzés során automatikusan létrejöttek a csoportok. A csoportok bezárásával rejtsük el az adatokat, és csak a statisztikai eredményeket jelenítsük meg!
2. Van-e összefüggés a nyelvválasztás és a tagozat megválasztása között? A fiúk vagy a lányok szerepeltek jobban a szóbeli beszélgetésen? Keressünk választ részösszegképzéssel és kimutatással egyaránt!
3. Nyissuk meg a köznevelési intézmények adatbázisát, és
   a) foglaljuk csoportba az intézmények székhelyére, illetve a fenntartóra vonatkozó adatokat!
   b) részösszegképzéssel határozzuk meg a köznevelési intézmények számát megyénként, azon belül járásonként!
   c) határozzuk meg, hány óvoda, általános iskola, illetve gimnázium működik az egyes megyékben!

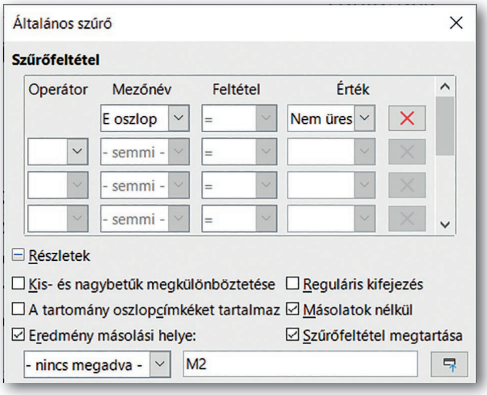
Feltételtől függő számítások

22. példa: A tanult nyelvek kigyűjtése és összegzése

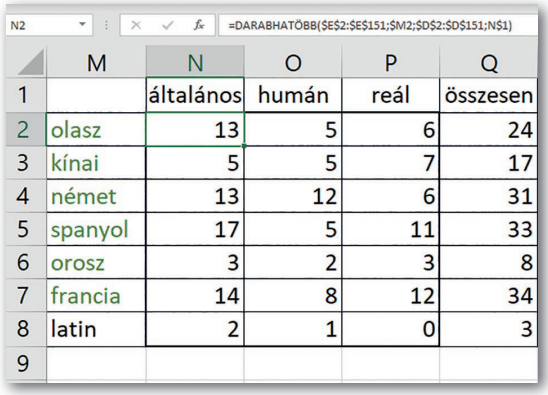
A második idegen nyelv oktatásának megszervezése az iskolában mindig komoly feladat: az iskola szeretné minden tanuló számára az elsőnek választott nyelvet biztosítani (*Nyelv1* mező), ugyanakkor a csoportok csak bizonyos létszám fölött indíthatók. A csoportok tervezéséhez gyűjtsük ki az elsőnek választott nyelvek listáját (minden nyelv csak egyszer szerepeljen benne), majd számoljuk meg, hogy tagozatonként hányan választották azokat!

A nyelvek listájának kigyűjtésére a táblázatkezelő programok különböző megoldásokat kínálnak. A Microsoft Excelben például erre külön parancs létezik. Másoljuk át az *E2:E151* tartomány adatait az *M2:M151* tartományba, majd meghagyva a tartomány kijelölését, indítsuk el az *Adatok* > *Ismétlődések megszüntetése* parancsot!

LibreOffice Calc esetén a feladatot szűréssel oldhatjuk meg. Jelöljük ki az *E2:E151* tartományt, és indítsuk el a szűrést az *Adatok* > *Több szűrő* > *Általános szűrő* menüponttal! A megjelenő ablakban írjuk elő, hogy az *E* oszlop *nem üres* értékeit szeretnénk szűrni, az alsó részen pedig adjuk meg, hogy a szűrt adatokat *másolás nélkül,* az *M2*-es cellától kezdve helyezze el a program!



► Az ismétlődések eltávolítása szűréssel (LibreOffice Calc)



	M	N	O	P	Q
1		általános	humán	reál	összesen
2	olasz	13	5	6	24
3	kínai	5	5	7	17
4	német	13	12	6	31
5	spanyol	17	5	11	33
6	orosz	3	2	3	8
7	francia	14	8	12	34
8	latin	2	1	0	3
9					

► Az egyes nyelveket választók száma tagozatonként (Microsoft Excel)

Másoljuk az 1. sorba a mintának megfelelően a tagozatok nevét, és határozzuk meg képlettel, hogy az egyes nyelveket hányan választották tagozatonként! Például a következő másolható képletet írhatjuk az *N2*-es cellába:

```
=DARABHATÖBB($E$2:$E$151;$M2;$D$2:$D$151;N$1)
```

Végül összegezzük a *Q* oszlopban, hogy az iskolában összesen hányan választották az egyes nyelveket, és emeljük ki feltételes formázással, zöld színnel azoknak a nyelveknek a nevét, amelyeket iskolai szinten legalább heten választottak! A feltételt az *M2:M8* tartomány kijelölése után például a következő képlettel adhatjuk meg: `=Q2>=7`.

23. példa: Adatbázis-kezelő függvények

Meghatározott feltételeknek megfelelő adatok összegzésére (átlaguk, maximumuk, minimumuk stb. meghatározására) többféle módszerrel találkoztunk. Az első és legegyszerűbb

megoldás az volt, hogy a szükséges adatokat logikai függvények alkalmazásával kigyűjtöttük egy segédoszlopba, majd azokat a megfelelő statisztikai függvénnyel összesítettük. A folyamatot egyszerűsítette a feltételes statisztikai függvények bevezetése („HATÖBB” függvények). A nagy adathalmazok használatánál megismerkedtünk a képletek beírását nem igénylő részösszegképzéssel és a kimutatással. Ezúttal kitekintünk egy újabb megoldásra: az adatbázis-kezelő függvényeket mutatjuk be.

A nyelvi adatokat elemezve felmerült, hogy a reál tagozat részére szervezzenek külön német nyelvi csoportot. Mivel a tagozathoz kijelölt matematikatanár német szakos is, így ezt a csoportot az igazgató neki szánta. A tanár szeretné megtudni, hogy azok a tanulók, akik a reál tagozatra jelentkeztek, és vagy első, vagy második helyen a német nyelvet választották, milyen eredményeket értek el a felvételi során matematikából, illetve hogyan szerepeltek a szóbeli vizsgán. Az adatokat szeretné összehasonlítani a reál tagozatra jelentkezők hasonló adataival is.

Az első vagy második helyen német nyelvet választó tanulók adatait a „HATÖBB” függvények egyszerű alkalmazásával nem tudjuk elemezni, mivel esetükben a német szó vagy a *Nyelv1,* vagy a *Nyelv2* mezőben szerepel. Az ilyen és akár ennél sokkal bonyolultabb feltételek kezelésére vezették be az adatbázis-kezelő (vagy röviden „AB”) függvényeket. Az „AB” függvények esetében ugyanis a feltételt – az irányított szűrésnél már megismert – szűrőtartomány segítségével adhatjuk meg. A szűrőfeltétel ezúttal:

```
Tagozat="reál" és (Nyelv1="német" vagy Nyelv2="német")
```

Mivel a reál tagozatra vonatkozó feltétel mindkét nyelvre igaz, és a szűrőfeltételt soronként a *vagy* logikai művelet köti össze, a feltételt így fogalmazhatjuk át:

```
(Tagozat="reál" és Nyelv1="német") vagy
(Tagozat="reál" és Nyelv2="német")
```

	M	N	O	P	Q	R	S
1		**Tagozat**	**Nyelv1**	**Nyelv2**			
2		reál	német				
3		reál		német			
4							
5		németes		teljes tagozat			Az *N* oszlopban szereplő képletek:
6		*matematika*	*szóbeli*	*matematika*	*szóbeli*		
7	fő	20	20				=AB.DARAB2(A1:J151;"Kód";N1:P3)
8	átlag	41,55	18,65				=AB.ÁTLAG(A1:J151;"Matematika";N1:P3)
9	legjobb	50	25				=AB.MAX(A1:J151;"Matematika";N1:P3)
10	legrosszabb	28	11				=AB.MIN(A1:J151;"Matematika";N1:P3)

▶ A reál tagozaton német nyelvet választók adatai összehasonlítva a teljes reál tagozat adataival

Hozzuk létre a fenti szűrőfeltételt az *N1:P2* tartományban, majd alakítsuk ki alatta a táblázatot (egyelőre adatok nélkül) a mintának megfelelően! Érdemes a mezőneveket átmásolni, mivel pontatlan mezőnév esetén az „AB” függvények nem működnek.

Az „AB” függvények használatához általában három paraméter szükséges. Az első paraméter az adatokat tartalmazó tartomány, az *adatbázis,* amely esetünkben az *A1:J151.* A második, hogy ennek melyik mezőjére kell alkalmaznunk az adott függvényt. Célszerű ilyenkor a mező nevét beírni, például *"Matematika"*, de helyette megadhatjuk a mező cellahivatkozását *(G1)* vagy a mező sorszámát (7). A harmadik paraméter a szűrőtartományt

megadó *kritérium,* amely most az *N1:P2*. A függvények neve megegyezik a megfelelő statisztikai függvényekkel, csak kiegészül az AB előtaggal: **AB.DARAB, AB.DARAB2, AB.SZUM, AB.ÁTLAG, AB.MAX, AB.MIN** stb.

A fentiek alapján az *N7*-es cellában például a következő képlettel számolhatjuk meg, hány nem üres mező van a feltételt kielégítő rekordok esetén a *Kód* mezőben:

```
=AB.DARAB2(A1:J151;"Kód";N1:P3)
```

A legjobb szóbeli eredményt pedig az *O9*-es tartományban a következő képlet adja:

```
=AB.MAX(A1:J151;"Szóbeli";N1:P3)
```

Az „AB” függvények közé tartozik, az **AB.MEZŐ** függvény is, amellyel gyakran kiválthatók a keresőfüggvények (például az `INDEX` és a `HOL.VAN` páros). Határozzuk például meg, melyik felvételiző érte el a legmagasabb pontszámot szóbelin!

A szűrőtartományt bővítsük tovább az *N12:Q14* tartományban – az ábrának megfelelően – a szóbelin elért legnagyobb pontszámmal (*Q13*-as és *Q14*-es cellába az előbbi képletet írjuk)! A legjobb szóbeliző nevét ekkor a következő képlet adja. (A képlet hibát jelez, ha nincs az adott feltételnek megfelelő érték, vagy több ilyen is van.)

```
=AB.MEZŐ(A1:J151;"Név";N12:Q14)
```

	N	O	P	Q	R	S
12	**Tagozat**	**Nyelv1**	**Nyelv2**	**Szóbeli**		
13	reál	német		25		Q13: =AB.MAX(A1:J151;"Szóbeli";N1:P3)
14	reál		német	25		
15						
16	A legmagasabb pontszámot érte el:			Papszt Ince		Q16: =AB.MEZŐ(A1:J151;"Név";N12:Q14)

► A legjobb szóbeli eredményt elérő tanuló a reál tagozaton német nyelvet választók közül

Kérdések, feladatok

1. Példánkban az `AB.MEZŐ` alkalmazásánál miért nem elegendő csak a szóbeli pontszámot megadni, miért kell azzal az eredeti kritériumot bővíteni?
2. Vajon mi történik, ha a kritériumtartomány üres sort tartalmaz? Válaszunkat indokoljuk!
3. Fejezzük be a feladatot a *P7:Q10* tartományba írt képletek megadásával! Itt már elegendőek ugyan a „HATÖBB” függvények, de a feladatot oldjuk meg nemcsak azokkal, hanem az „AB” függvények alkalmazásával is!
4. Csoportmunkában, a táblázatkezelő program alkalmazásával oldjuk meg a következő problémákat!
   a) Megadható-e úgy a tanulók nyelvi beosztása, hogy induljon csoport minden elsőnek választott nyelvből (a csoportlétszám minimum 8 fő)?
   b) Vizsgáljuk meg, hogy másodikként milyen nyelveket jelöltek meg a tanulók! Megadható-e úgy a tanulók nyelvi beosztása, hogy minden választott nyelvből induljon csoport?
   c) Amennyiben az iskolának takarékoskodnia kell, akkor milyen beosztásban indítható el a legkevesebb nyelvi csoport, ha a maximális csoportlétszám 24 fő?

Adatbázis-kezelés

Az adatbázis-kezelés fogalmai

Civilizációnk fejlődése során egyre több tudást halmozott fel, közben egyre összetettebben működő társadalmat alakított ki. A kereskedelem, az adózás megjelenésével számtalan adat keletkezett, amelyeket gyakran táblázatos formában rögzítettek. Táblázatba foglalták az égitestek mozgására vonatkozó megfigyeléseket és az anyakönyvi eseményeket is.

Adót nem fizetett mindenki, a tudomány régen csak kevesek kiváltsága volt, de az egyházi, majd állami anyakönyvekbe gyakorlatilag mindenki belekerült. (Az Osztrák–Magyar Monarchiában 1895-ben tértek át állami anyakönyvezésre.) Születéskor, kereszteléskor rögzítették a dátumot, a gyermek és a szülők nevét. Bekerült a házasságkötés és a halálozás időpontja is. Ezek az anyakönyvek képezik az alapját a családfakutatásoknak. Minden rendelkezésre áll, mégsem könnyű néhány száz évre visszamenőleg összeállítani egy családfát. Nehézséget okoznak a névelírások és a papíralapú tárolás fizikai széttagoltsága.

Utóbbit megoldhatnánk, ha a különböző településeken, levéltárakban tárolt anyakönyvek tartalmát pontosan egyező szövegezéssel egy „könyvbe", azaz egyetlen szöveges állományba gyűjtenénk. Ezzel a keresési idő jelentősen csökkenne, de az adatrögzítési hibák nem tűnnének el. Egy elírt név továbbra is komoly fejtörést okozna, az egyező nevek szintén megnehezítenék a múlt megismerését. Jól látható, hogy nem elegendő pusztán egy új adathordozót használnunk, mert azzal ugyanazt csináljuk, csak gyorsabban, az igazi előrelépéshez szemléletet is kell váltanunk.

1. példa: A *diákok* adatbázis megtervezése

Nézzük meg egy példán lépésről lépésre, hogy miképpen alakíthatjuk át az egyszerű szöveges lejegyzést a hatékony adatfeldolgozás érdekében!

A szövegrészletben három diákról egy-egy mondat szerepel. Minden diákról ugyanazokat az ismereteket tartalmazzák a mondatok: hány éves, hol született, milyen nyelvet tanul, melyik iskolába jár, hány diák jár abba a suliba.

Nóra 13 éves, Budapesten született, angolul és németül tanul, a Piros iskolába jár, amely Budapesten található, és 652 diákja van.

***Pista** a neve annak az **angol** nyelvet tanuló, **11** éves diáknak, akinek születési helye **Sopron**, a **Piros** iskolába jár, amely **Budapest** városában található, és amelynek **652** diákja van.*

A Szolnokon született Bea 12 éves, a Kék iskolába jár, amely Szegeden található, és 541 diákja van.

Annyi mondatot írhatnánk, ahány diákot ismerünk. Milyen problémát rejt az adatrögzítésnek ez a módja? Az adatok közötti keresést nehezíti az eltérő megfogalmazás. Sokat segítene a rögzített szórend, hiszen tudnánk, hogy hova kell fókuszálni, ha például a budapesti születésűeket keressük. Néhány mondat után érezzük, hogy azokat a szavakat,

amelyek minden mondatban szerepelnek, fárasztó és felesleges leírni – régen sem tették, macskakörmözést használtak helyettük. Vegyük észre, hogy ezen ismétlődő szavak, kifejezések egy másik szóval alkotnak egy párt, például *név – Pista, születési hely – Sopron*. A szópárok egyik tagja egy tulajdonság, a másik pedig a tulajdonság adott diákhoz tartozó értéke. A második mondatban a tulajdonságot aláhúzással, az értéket félkövéren ki is emeltük.

A mondatok helyett táblázatot készíthetünk, ha az aláhúzott szavakat, kifejezéseket kiemeljük a fejlécbe, a félkövéren írtakat pedig a sorokon belül a megfelelő oszlopban tüntetjük fel.

név	**éves**	**nyelvek**	**születési hely**	**iskolanév**	**város**	**diákok száma**
Nóra	13	angol, német	Budapest	Piros	Budapest	652
Pista	11	angol	Sopron	Piros	Budapest	652
Bea	12		Szolnok	Kék	Szeged	541
…	…		…	…	…	…

Ezzel az áttekinthetőség sokat javult. Az első sorban a tulajdonságok elnevezése szerepel, ezeket **mezőnévnek** hívjuk. A mezőnév alatt a megfelelő tulajdonság egy-egy lehetséges értéke van megadva. A táblázat soraiban egy-egy diákhoz tartozó, tehát egymással összefüggésben – idegen szóval relációban – lévő **értékek** olvashatók. Az adatsorok neve szakszóval **rekord**. Ha egy táblázat minden oszlopában azonos szerepű értékek találhatók, az egyes sorok értékei pedig összefüggésben vannak egymással, **adattábláról** beszélünk.

A táblázatos forma logikus és áttekinthető. Érdemes azonban végiggondolni, hogy praktikusan választottuk-e meg a tartalmát. Vajon mennyire időtálló a tartalma? Néhány problémát könnyen felfedezhetünk:

- Nórából, Pistából még a Piros iskolában is sok lehet, hát még az országban. Segítene, de teljes mértékben még az sem oldaná meg a gondot, ha a vezetéknevüket ismernénk.
- Az idő múlásával változik az életkor, így szinte napról napra követnünk kellene, hogy kinek volt éppen születésnapja, kinek az életkorát kell módosítanunk.
- Az iskolaváltások megváltoztatják a létszámot, ezért ha Pistáék Szegedre költöznek, és ő a budapesti Piros iskola helyett a szegedi Kék iskolában folytatja tanulmányait, a 652 + 541 sorban kell módosítani a diákok számát. Hasonló probléma adódik akkor is, ha a nyolcadikosok helyét az elsősök veszik át szeptemberben.

E problémákat megoldhatjuk, ha

- rögzítjük a diákok oktatási azonosítóját is, hiszen az mindenkinek egyedi;
- nem az életkort, hanem a születési dátumot (esetleg évet) tartjuk nyilván;
- az egyes iskolákhoz tartozó diákok számát nem rögzítjük, mert – ha tényleg az ország összes diákjának adatait tároljuk – minden olyan esetben meg tudjuk számlálni, amikor szükség van rá.

A javított táblázat:

oktatási azonosító	név	nyelvek	születési dátum	születési hely	iskolanév	város
71234567890	Nóra	angol, német	2008.05.01.	Budapest	Piros	Budapest
71234564534	Pista	angol	2010.03.16.	Sopron	Piros	Budapest
71234553463	Bea		2009.06.12.	Szolnok	Kék	Szeged
…	…		…	…	…	…

Három problémát is kiküszöböltünk, de vajon tökéletes-e? Korántsem, hiszen ha az iskola a szomszéd településre költözne, akkor az összes oda járó diáknál meg kellene változtatni ezt a tulajdonságot. A fenti táblázat tehát nemcsak a diákokról tartalmaz adatokat, hanem az iskolákról is. Jobb, ha egy iskola települését, címét nem diákonként jegyezzük meg, hanem csak egyszer, az iskola nevéhez kapcsoltan. Rögzítsük külön táblázatban a diákok és az iskolák adatait!

Ahogy a diákoknál a név, önmagában az iskola neve sem határozza meg az iskolát, így az iskolához is adjunk hozzá egy egyedi értéket, azonosítót! Legyen ez az OM-azonosító! Ahhoz, hogy tudjuk, melyik diák melyik iskolába jár, rögzítsük a megfelelő OM-azonosítót a diák tulajdonságaként!

A nyelvek esetén érezhetünk még problémát, hiszen Nóra két nyelvet is tanul, Beánál pedig egy sem szerepel. Ha a diákokat tanult nyelv szerint keressük, akkor a többértékű mezők nehezíthetik a dolgunkat. Ha egy mezőnél több érték is szerepelhet, vagy előfordul üres érték, akkor javasolt külön táblázatban vagy táblázatokban tárolni ezeket az adatokat. Nézzük, hogy miképpen!

Diák táblázat

oktatási azonosító	név	születési dátum	születési hely	iskola
71234567890	Nóra	2008.05.01.	Budapest	789123
71234564534	Pista	2010.03.16.	Sopron	789123
71234553463	Bea	2009.06.12.	Szolnok	789225
…	…	…	…	…

Iskola táblázat

OM-azonosító	iskolanév	város
789123	Piros	Budapest
~~789123~~	~~Piros~~	~~Budapest~~
789225	Kék	Szeged
…	…	…

Nyelv táblázat

azonosító	név
1	angol
2	francia
3	német
…	…

Nyelvtanulás táblázat

azonosító	diák oktatási azonosító	nyelvazonosító
1	71234564534	3
2	71234567890	1
3	71234567890	3
…	…	…

Gondoljunk vissza a példamondatainkra! A „típusmondatot” könnyen átírtuk adattáblává, de a négy adattáblás végeredményig sok lépésből álló út vezetett. Az alábbiakban igyekszünk általánosabban megfogalmazni a lépéseket:

- Ne tároljunk olyan értéket, amely a többi tárolt adat alapján meghatározható (diákok száma)!
- Ne tároljunk olyan értéket, amely minden beavatkozás nélkül, automatikusan változik (életkor), cseréljük le olyanra, ami állandó!
- Minden adattáblában legyen egy azonosító szerepű tulajdonság (oktatási azonosító, OM-azonosító), akár egy sorszám (nyelv, nyelvtanulás), amely egyedi, ezáltal meghatározza a sor többi adatát!
- Ne legyen olyan oszlop, amelynek celláiba több értéket is bejegyzünk (nyelv), és lehetőség szerint kerüljük el az olyan oszlopokat is, ahol a cellákban (gyakran) nem szerepelnek értékek!

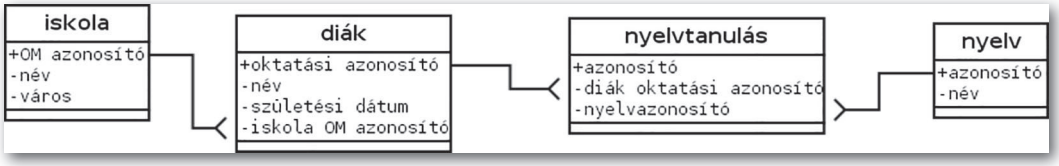
Ha valaki a fenti pontok szerint alakítja át a kiindulási táblázatát, akkor egy több adattáblából álló, jól használható struktúrához jut, amelyet **relációs adatmodellnek** is nevezik. (A reláció itt a soronként egymással kapcsolatban álló, összefüggő értékekre utal.) A szakemberek ezt az átalakítási folyamatot **normalizálásnak** nevezik. A normalizálás lépései a fenti tanácsok alapján kialakított struktúrát eredményezik. Mivel ebben a struktúrában az **adattáblák logikai kapcsolatban állnak egymással, adatbázisról** beszélhetünk. (Lásd a 10. évfolyamos tankönyvet.)

Az adatbázis általánosságban rendszerezetten tárolt adatok összessége, amely nemcsak relációs adatmodell alapján alkotható meg, hanem lehet hálós vagy hierarchikus is. (Olvassunk utána, mit jelent az adatbázis normalizálása!)

Néha előfordul, hogy praktikus okokból a fenti javaslatok egyikét-másikát figyelmen kívül hagyják, de mi ezt csak indokolt esetben tegyük. Az adatbázisok gyakran nem egy, nem négy, hanem több tucat táblából állnak, így nem könnyű jól használható normalizált adatbázist készíteni. A fenti példának nem az a célja, hogy profi szakemberré váljunk, hanem hogy könnyebben megértsük az általunk használt adatbázisok felépítését, és ezáltal hatékonyabban oldjuk meg a hozzájuk kapcsolódó feladatokat.

Relációs adatmodell kialakítása

A fenti táblázatokban az adatok csak példák, azt szolgálják, hogy könnyebben megértsük az egyes oszlopok szerepét. Aki rutinos az adatbázis-kezelésben, sokkal tömörebben is rögzítheti az adatbázis szerkezetét.



▶ A *diákok* adatbázis szerkezete (A rajz a Dia diagramszerkesztő programmal készült.)

A fenti ábra – mivel az adattáblák közötti kapcsolatokat is mutatja – az adatokat is tartalmazó változatnál több információt hordoz. Az azonosító szerepű tulajdonságokat a + karakterrel jeleztük. Ezt szokás **kulcsként** is nevezni. Látható, hogy az adattáblák közötti **kapcsolatok** a kulcsból indulnak, és a másik táblában, a kapcsolat túlsó végén ezek az elemek megjelennek egyszerű tulajdonságként. Utóbbit **idegen kulcsnak** nevezzük. A táblázatban az azonosító oszlopában minden érték pontosan egyszer szerepel, az idegen kulcs oszlopban pedig többször is előfordulhat. Ezt az összekötő vonal elágazásával jelezzük.

Vegyük észre, hogy az adattáblák kapcsolatának meghatározásánál azt kell eldöntenünk, hogy melyik azonosítóját kell beírnunk a másik táblába tulajdonságként.

Iskola–diák: Egy iskolába járhat-e több diák? Egy diák járhat-e több iskolába? Az első kérdésre igen a válasz, a másodikra nem. Ezért a diáktáblába kell felvennünk tulajdonságként az *iskola* tábla azonosítóját. A kapcsolatot megteremtő érték az egyik táblában csak egyszer fordulhat elő, a másikban többször is, ezért ezt **egy a többhöz kapcsolatnak** nevezzük.

Diák–nyelv: Egy diák tanulhat-e több nyelvet? Egy nyelvet tanulhat-e több diák? Mindkét kérdésre igennel válaszolunk. Ilyen, több a többhöz kapcsolat a relációs adatbázisban nem szerepelhet, ezért egy új táblát kell készítenünk, amelybe a két másik tábla azonosítóiból alkotott párokat jegyzünk be. Ezt a táblát **kapcsolótáblának** hívjuk.

Feladatok

1. Készítsünk adatbázis-szerkezetet a következő típusmondatokhoz!
   a) Svájcban a fizetőeszköz a frank, az ország államformája köztársaság, jelentősebb városai Zürich, Bern, Bázel.
   b) Az 1825-ben, Komáromban született Jókai Mór írta *A kőszívű ember fiai* és *Az arany ember* című regényeket is.
   c) Komáromi József Debrecenben született, jelenleg Szolnokon lakik, a következő e-mail-címeket használja: kjoco@komaromi.hu, komaromi.jozsef@munkahely.hu.
2. Milyen kapcsolatban állhatnak az alábbi táblák, melyikbe kell bejegyeznünk idegen kulcsként a másik azonosítóját?
   a) város és ország
   b) film és színész
   c) költő és költemény
   d) bolt és vásárló

Adatbázis a számítógépen

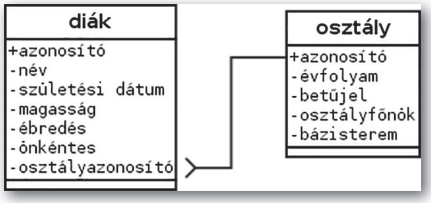
Az előző leckében megismertük, hogy egy adathalmaz tárolásához hogyan alakítunk ki **relációs adatmodellt.** Most megnézzük, milyen módon lehet a kialakított szerkezetben eltárolni az adatokat a számítógépen.

2. példa: Az *iskola* adatbázis megtervezése

Tekintsük az alábbi típusmondatot!

Nagy Nóra 2008. 05. 01-jén született, 1,51 m magas, minden tanítási napon 6:15-kor kel, vállal önkéntes segítői munkát, a 7. A osztályba jár, amelynek osztályfőnöke Virágh Zoltán, bázistermük a 231-es.

Ebből a képen látható szerkezetű adatbázist alkottuk.



▶ Az *iskola* adatbázis szerkezete

Próbáljuk megválaszolni az alábbi kérdéseket!

- Miért választottuk kétfelé az adatokat?
- Milyen szerepet tölt be a diáktáblában található osztályazonosító mező?
- Miért tároljuk két részletben az osztály megnevezésének évfolyam és betűjel részét?
- Miért nem az osztály megnevezése (7. A) azonosítja Csaba osztályát?
- Mikor döntenénk egy külön *tanár* vagy *terem* tábla létrehozása mellett? Hogyan befolyásolná a fenti rajzot?

Ha e szerint a relációs adatmodell szerint vetjük papírra az adatokat ténylegesen tartalmazó táblázatokat, akkor csak az oszlopok szélességére kell ügyelnünk. Ha számítógépes tároláson gondolkodunk, akkor nem hagyhatjuk figyelmen kívül **az adatok típusát** sem, hiszen az határozza meg, hogy milyen műveleteket végezhetünk rajtuk. A programozás során már megtapasztaltuk, hogy nem mindegy, szövegként vagy számként tárolunk-e egy adatot, és hogy számként egész vagy valós típust választunk-e.

Vegyük sorra, hogy az adatmodell egyes mezőihez milyen típust praktikus – vagy éppen kötelező – használni!

A diák *azonosítójaként* továbbra is az oktatási azonosítót használjuk. Ez minden esetben pontosan 11 számjegyből áll. Természetesnek tűnne, hogy szám típusúként tüntetjük fel, mégis szöveges típust kell alkalmazni. Gondoljunk arra, milyen műveletet végezhetünk ezzel az értékkel! Nincs olyan feladat, amelynél a számokon értelmezett matematikai műveleteket használhatnánk. Egy másik fontos észrevétel, hogy előfordulhat az ehhez hasonló azonosítók között olyan is, amely 0-s karakterrel kezdődik. (Az oktatási azonosító mindig 7-es számjeggyel kezdődik.) A vezető nullák tárolására szám esetén nincs mód. Tehát az oktatási azonosítót 11 karakteres szövegként adjuk meg.

A *név* tárolását egy mezőben végezzük. Ez nem magától értetődő, ugyanolyan jó megoldás lenne különválasztva tárolni a vezeték- és az utónevet. (Egyes esetekben még a családnevet megelőző dr., id., ifj. rövidítést is külön tárolják.) Ha nincs a felhasználás céljához tartozó indok a szétválasztásra, így sem követünk el hibát. Típusa szöveg. Ha az általunk használt programban szükséges megadnunk a hosszát, keressük meg az általunk ismert leghosszabb nevet, növeljük meg a hosszát az ötödével, és azt használjuk mezőhosszként! Más esetekben is érdemes hasonlóan túlbecsülni a méretet, hiszen előfordulhatnak az általunk ismertnél nagyobb helyet igénylő adatok.

A *születés dátumát* és az ébredés idejét természetesen dátum, illetve idő típusként rögzíthetjük.

A *magasság* méterben mért értéke nyilvánvalóan szám, mégpedig valós szám.

Az *önkéntesség* oszlopába igen vagy nem értéket lehet beírni, amelyet helyettesíthetünk az igaz/hamis szavakkal, de akár a 0 vagy 1/-1 értékekkel is. Ez logikai típus.

Nézzük meg, hogy az általunk tanult programozási nyelven az Igaz logikai érték és az 1 számérték egyenlő-e!

Az alábbi táblázatok alsó sorában feltüntettük, hogy milyen típust választhatunk a tároláshoz.

Diák tábla

azonosító	név	születési dátum	magasság	ébredés	önkéntes	osztály-azonosító
71234567890	Nagy Nóra	2008.05.01.	1,51	6:15	igen	23
…	…		…	…	…	…
szöveg (11)	szöveg (30)	dátum	valós szám	idő	logikai	egész szám

Osztály tábla

azonosító	évfolyam	betűjel	osztályfőnök	bázisterem
23	7	A	Virágh Zoltán	231
…	…		…	…
egész szám / számláló	egész szám	szöveg (1)	szöveg (50)	szöveg (10)

Próbáljunk válaszolni az alábbi kérdésekre!

- Miért tartunk fent nagyobb szélességű oszlopot az osztályfőnök neve számára, mint a diák neve számára?
- Miért szöveg típusú a bázisterem?
- Milyen széles oszlopot használnánk iskolánkban a bázisterem tárolására?

Típusok

A fenti példában csak az a néhány típus fordult elő, amely minden, középiskolában elterjedten használt programban elérhető, nincs is többre szükségünk a tanulmányaink során. E szoftverek mindegyike alkalmas arra, hogy az adatbázis-kezelési feladatokat – a programozáshoz hasonlóan – szöveges utasítások kiadásával oldjuk meg.

Az alábbi táblázat a parancssorban használt típusneveket tartalmazza.

	MS Access*	BASE	MySQL	PostgreSQL
szöveg	**varchar (n)** – *rövid szöveg*	varchar (n)	varchar (n)	character varying
egész szám	**integer** – *szám, hosszú egész*	integer	integer	integer
valós szám	**double** – *szám, dupla*	double	double	double precision
dátum	**date** – *dátum/idő*	date	date	date
idő	**time** – *dátum/idő*	time	time	time
logikai	**logical** – *igen/nem*	boolean	tinyint(1)	boolean
számláló	**counter** – *számláló*	integer identity	integer auto_ increment	serial

*A dőlt betűs típusneveket akkor használjuk, ha az adattáblát űrlappal készítjük.

Látható, hogy a típusnevekben alig van különbség. A később használt parancsok felépítésében sem találnánk túl sok eltérést a középiskolai tanulmányok szintjén előkerülő feladatokban.

Az MySQL és a PostgreSQL adatbázisai egy kiszolgálón találhatók, és a felhasználók sokféle programon keresztül, szövegesen megfogalmazott, SQL (Structured Query Language, azaz strukturált lekérdezőnyelv) nyelvű utasításokkal kezelik őket. Az SQL-nyelv alkalmas az adatbázis létrehozására, az adatok manipulálására, lekérdezésére és a hozzáférés szabályozására.

Az Access és a Base programok általában helyi gépen tárolt adatbázist használnak. A feladatok megoldásához nem szükséges parancsokat gépelnünk, de az adatlekérdező és -manipuláló műveleteket SQL-nyelven is megadhatjuk.

Ebben a könyvben az Access program segítségével mutatjuk be a feladatok megoldását, mert a legtöbb esetben egy jól áttekinthető felületen, kevés gépeléssel, a lényegre koncentrálva dolgozhatunk. A megoldást szemléltető képernyőképeket is ebben a programban készítettük. (A Microsoft Access program 2019-es verzióját használtuk, de ez alig tér el a 2013-as és 2016-os verzióktól.)

Adatbázis létrehozása

Hozzuk létre az általunk használt adatbázis-kezelő rendszerben az *iskola* adatbázist!

A számítógépen létrehozott adatbázisba minden esetben beleértjük az adattáblákat – az azokat alkotó mezők nevével, típusával, a kulcs megjelölésével –, a köztük lévő kapcsolatokat és a bennük tárolt adatokat is.

Az Access programban az adatbázis állományában tároljuk

- az adatfelvitelhez készített felületeket (űrlapokat),
- az adatok lekérdezését és manipulálását biztosító utasításokat (lekérdezéseket),
- az esztétikus megjelenítést, nyomtatást segítő felületeket (jelentéseket).

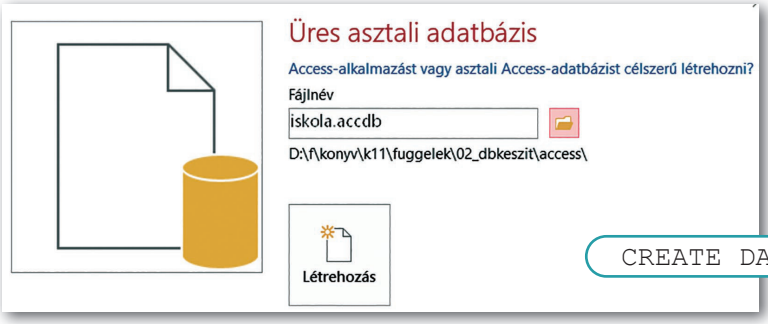
Az alábbiakban az *iskola* adatbázis létrehozását mutatjuk be. A grafikus felületen elvégzett műveletek mellett feltüntetjük az SQL-nyelvi megfelelőt is, amely a típusok nevének megfelelő módosításával a többi programban is használható.

Az Access program, az irodai programcsomag többi elemével szemben, új adatbázis készítésekor azonnal kikényszeríti a mentést. Így a későbbiekben magát az adatbázist nem kell rendszeresen mentenünk, az adatvesztés veszélye kisebb, mint például a táblázatkezelőben.

A tankönyvben a táblaneveket és a mezőneveket, valamint a lekérdezések, a jelentések és az űrlapok nevét az egyszerűbb kezelhetőség érdekében egybeírva, néhol ékezetek nélkül adjuk meg.

3. példa: Az *iskola* adatbázis létrehozása

Hozzuk létre az adatbázist!




```
CREATE DATABASE iskola;
```

▶ Az *iskola* adatbázis létrehozása

Hozzuk létre a *diák* és az *osztály* táblákat!

Ne feledkezzünk meg a kulcs beállításáról sem! Az egyes mezők esetén figyeljük meg a mezők listája alatt az általános tulajdonságokat, ahol a szöveges mező hosszát, valamint a szám valós vagy egész voltát állíthatjuk be.


```
CREATE TABLE diak (
  azonosito VARCHAR(11) PRIMARY
  KEY,
  nev VARCHAR(30),
  szuldat DATE,
  magassag DOUBLE,
  ebredes TIME,
  onkentes LOGICAL,
  osztalyazonosito INTEGER);
```

▶ A *diák* tábla létrehozása

A táblatervező nézetében az elsődleges kulcsot utólag is megjelölhetjük az eszköztár vagy a helyi menü segítségével:


```
ALTER TABLE diak ADD PRIMARY KEY(azonosito);
```

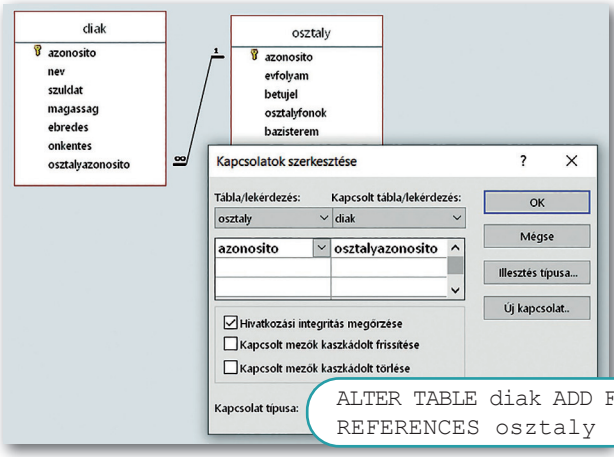
► Azonosító létrehozása a *diák* táblában

Az *osztály* táblában az elsődleges kulcs egy sorszám, amelyet számláló típusként kell rögzítenünk:


```
CREATE TABLE osztaly (
  azonosito INTEGER
  AUTO_INCREMENT PRIMARY KEY,
  evfolyam INTEGER,
  betujel VARCHAR(1),
  osztalyfonok VARCHAR(50),
  bazisterem VARCHAR(10));
```

► Az *osztály* tábla létrehozása

A táblák közötti kapcsolatot az *Adatbáziseszközök* rész *Kapcsolatok* pontját választva tudjuk beállítani. A hivatkozási integritás megőrzése biztosítja, hogy csak olyan osztályba rögzítsünk diákot, amelyik létezik is.




```
ALTER TABLE diak ADD FOREIGN KEY (osztalyazonosito)
REFERENCES osztaly (azonosito);
```

► Idegen kulcs létrehozása a *diák* táblában

Töltsük fel adatokkal a táblákat! Vigyázzunk arra, hogy a mintán látható 23-as azonosítójú osztályt csak a beszúró parancs segítségével lehet rögzíteni, a kézi adatfelvitel során a sorszámozás az 1-es értékkel kezdődik.

osztaly

azonosito	evfolyam	betujel	osztalyfonok	bazisterem	Hozzáadás
23	7	A	Virágh Zoltán	213	
24					
(Új)	0				

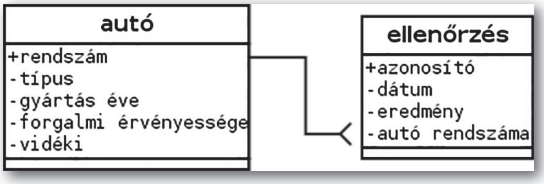
```
INSERT INTO osztaly (azonosito, evfolyam, betujel, osztalyfonok, bazisterem)
VALUES (23, 7, ,A', ,Virágh Zoltán', ,213');
INSERT INTO diak (azonosito, nev, szuldat, magassag, ebredes, onkentes,
  osztalyazonosito)
VALUES (,71234567890', ,Nagy Nóra', ,2008-05-01', 1.51, ,6:15:00', 1, 23);
```

▶ Adatbevitel az *osztály* táblába

Feladatok

Vizsgáljuk meg a következő típusmondatot: Az ABC-123 rendszámú, Kia típusú, 2013-ban gyártott autót, amelyet Budapesten használnak, 2018. 10. 08-án és 2020. 12. 13-án ellenőrizték, mindkétszer megfelelt.

A fenti mondathoz az adatbázis alábbi relációs adatmodelljét készítettük.



▶ A *taxi* adatbázis szerkezete

Oldjuk meg a következő feladatokat!

a) Milyen típusú és méretű mezőket kell használnunk az egyes táblákban?
b) Helyes-e az ábrán a kapcsolat iránya? Miért?
c) Hozzuk létre az adatbázist az általunk használt eszköz által biztosított grafikus felületen!
d) Állítsuk be az idegen kulcsot is!
e) Melyik táblába kell bevinnünk az első rekordot? Válaszunkat indokoljuk!
f) Vigyünk fel legalább két rekordot az *autó* táblába, és legalább öt rekordot az *ellenőrzés* táblába!
g) Töröljük az utoljára felvitt ellenőrzést, és vegyünk fel egy újabbat! Milyen azonosítót kapott az új rekord?
h) Próbáljunk meg felvenni olyan rendszámot az *ellenőrzés* táblába, amely nem szerepel az autó rekordjai között! Értelmezzük a hibaüzenetet!

Adatok importálása

Az előző leckében az *iskola* adatbázist mi hoztuk létre, és mi rögzítettünk benne néhány adatot. Célunk annyi volt, hogy megismerjük a különböző típusú adatok bevitelének módját és az adatbevitel sorrendjét. A legtöbb adatbázisba a valóságban is kézi adatbevitellel vagy legalábbis emberi közreműködéssel rögzítik az adatokat. Egy webáruház óriási kínálatát bizony árucikkenként viszik fel az alkalmazottak, és a vásárlók is minden megrendeléssel újabb és újabb rekorddal bővítik a táblák tartalmát. A tanulás során azonban nem hagyatkozhatunk kézi adatfelvitelre, mert nagyon sok időbe telne, mire akkora rekordszámú adattáblákat állítanánk elő, amelyeknél indokolt az adatbázis-kezelő használata.

A későbbiekben általában készen kapjuk az adattáblák feltöltéséhez szükséges adathalmazt, ritkábban pedig mi magunk állítjuk elő.

Adatbázis feltöltése szövegfájlból

4. példa: A *város* adatbázis megtervezése

Készítsük el az alábbi típusmondathoz tartozó adatbázist!

Hódmezővásárhely megyei jogú város, területe 487,98 km², népessége 43 311 fő, Csongrád-Csanád megyében fekszik, amelynek székhelye Szeged.

Természetesen átalakíthatjuk táblázatos formára a mondatot, de talán nem olyan bonyolult néhány kérdés megválaszolása után a relációs modell azonnali felrajzolása sem.

- Szerepel-e olyan adat, amely automatikusan, más esemény hatása nélkül megváltozik?
- Szerepel-e olyan adat, amely a többi adatból következik?

Mindkettőre nem a válasz.

- Hány „dologról” szól a mondat?

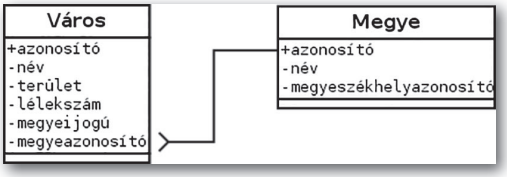
A városról és a megyéről szól. Tehát két táblát kell felrajzolnunk.

- Vajon a városnál kell-e felvennünk a megye azonosítóját idegen kulcsként, vagy a megyénél a városét?

Ehhez két kérdést kell feltennünk:

- Tartozhat-e egy megyéhez több város?
- Tartozhat-e egy város több megyéhez?

Ha csak az egyik kérdésre válaszolhatunk nemmel (itt a másodikra), akkor abba a táblába *(város)* kell bejegyeznünk a másik azonosítóját.



► A *városok* adatbázis szerkezete

A fentiekhez a képen látható adatbázis-szerkezetet választottuk:
Ha csak a rajzot látjuk, a két adattáblát összekötő vonal elágazásáról leolvasható, hogy egy megyéhez több város tartozik.

Határozzuk meg, hogy milyen típusúak az egyes mezők!

Az *azonosítók* típusa legyen **számláló,** mivel nincs olyan elterjedten használt tulajdonság, amely azonosítaná a várost vagy a megyét. (Ha a megyenév lett volna a kulcs, sok helyen kellett volna értéket módosítani, amikor 2020. június 4-én Csongrád megye nevét Csongrád-Csanádra változtatták.) A *nevek* legyenek **szövegesek,** méretüket pedig határozzuk meg a rögzíteni kívánt adatok alapján! A *lélekszám* és a *terület* **szám** típusú legyen, előbbi **egész,** utóbbi **valós.** A *megyeijogú* mezőben azt kell tárolnunk, hogy az adott település megyei jogú-e, vagy sem, így **logikai** típust kell választanunk. Az idegen kulcs típusát a másik tábla azonosítójának típusa határozza meg, azzal egyezőnek kell lennie.

Oldjuk meg az alábbi feladatokat a már megismert adatbázis-kezelő rendszerben!

- Hozzuk létre az adatbázist!
- Készítsük el a két táblát! (Ez grafikus felületen is megtehető, nem fontos SQL-utasításként.)
- Adjuk meg az idegen kulcsot! (Határozzuk meg a táblák közötti kapcsolatot!)
- Vigyünk be egy-egy adatsort a táblákba a mondatban olvasható adatok alapján!
- Vigyük be még egy megye és a megye néhány városának adatait!

Az adatbázist létrehozó SQL-utasításokat a források között `varosok.sql` néven helyeztük el.

A táblák létrehozása után válaszoljuk meg a következő kérdéseket!

- Miért a megye rögzítésével kellett kezdenünk?
- Mikor rögzíthetjük a *megye* táblában a *megyeszékhelyazonosító* értékét?
- Vajon megadhatjuk-e egy újabb idegen kulcsként a *megyeszékhelyazonosító* mezőt? Indokoljuk meg a választ! (Ha nem tudunk dönteni, próbáljuk ki, milyen hatása lesz, ha megadjuk.)

A végső cél természetesen az összes város és megye sémában szereplő adatainak rögzítése. Mivel háromszáznál is több város van, ezért az adatok összegyűjtése és begépelése időigényes. Ha az adattáblák tartalma szövegfájlban rendelkezésre áll, akkor az néhány lépésben beemelhető az adatbázisba.

A szükséges adathalmazt a Wikipédián megtaláljuk, így csak a megfelelő adatok táblázatba rendezése igényel munkát. Ezt végigcsinálhatjuk a következő példa alapján, vagy használhatjuk a források `varos.txt` és `megye.txt` állományait!

5. példa: Egy Wikipédia-oldal táblázattá alakítása

Látogassunk el a *https://hu.wikipedia.org/wiki/Magyarország_városai* lapra! (Ha éppen nem elérhető, használjuk a fejezethez tartozó állományok közül a `Magyarország városai – Wikipédia.htm` fájlt!)

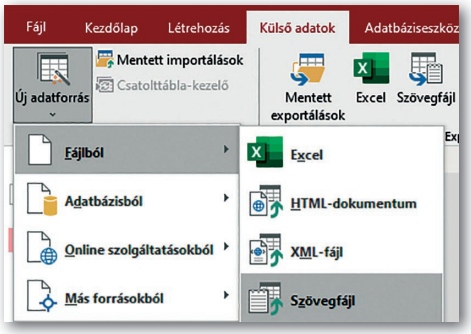
Az adatforrás két táblázattá alakításához alkalmazzuk a táblázatkezelés kapcsán tanultakat!

Egy lehetséges megoldás:

1. Másoljuk át a forrásban található táblázatot egy munkafüzet forrásmunkalapjára!
2. Töröljük a felesleges oszlopokat (név, típus, megye, lélekszám, terület maradjon)!
3. Vegyük fel az *azonosító* és a *megyeijogú* oszlopokat!
4. Az azonosító oszlopát töltsük fel számokkal!
5. A *megyeijogú* oszlopban megfelelő szűrés (szövegszűrő) után kézzel jegyezzük be az értékeket! (Használhatunk sorba rendezést vagy arra alkalmas függvényt is.)

6. A megyéket a *megye* oszlop adatainak felhasználásával vagy a megyeszékhelyre való szűréssel is előállíthatjuk. Ellenőrizzük, szerepel-e minden megye!
7. A városoknál a megyeazonosítókat határozzuk meg alkalmas függvénnyel!
8. A városokra és a megyékre vonatkozó adatokat másoljuk át két külön szövegfájlba (`varos.txt`, `megye.txt`)!

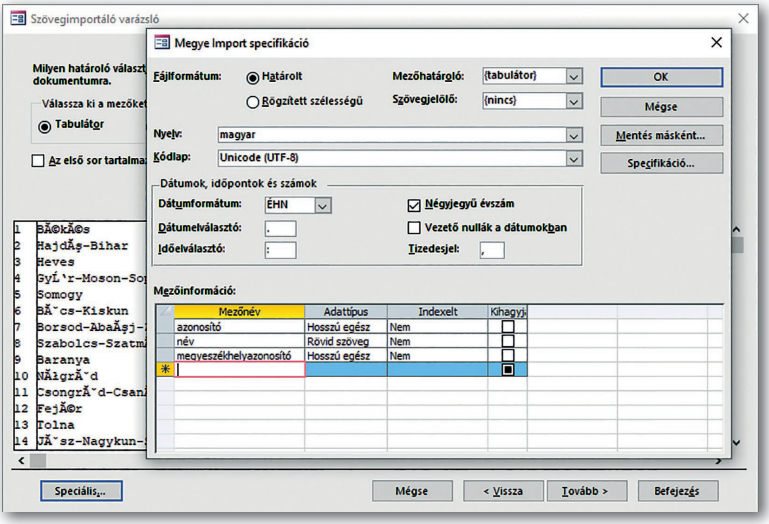
Az általunk előállított vagy a készen kapott szövegfájlok tartalmát helyezzük el a megfelelő adattáblákban! Ezt a műveletet **importálásnak** nevezzük.



► Importálás szövegfájlból

Az importálás során az alábbi lépéseket hajtsuk végre, de előtte ellenőrizzük, hogy a mezők sorrendje a fájlban és az adattáblában egyezik-e.

- Válasszuk a *Külső adatok* > *Új adatforrás* > *Fájlból* > *Szövegfájl* menüpontot, majd válasszuk ki a `megye.txt` fájlt!
- Mivel a tábla már létezik, ezért használjuk a *Rekordok másolatának hozzáfűzése* lehetőséget a megfelelő táblát választva!
- A megjelenő párbeszédablakban a *Speciális* gomb segítségével állítsuk be, hogy milyen mezőhatárolót használunk – ez általában tabulátor –, és adjuk meg a kódlapot, amely szinte mindig UTF-8.



► Az importált szövegfájl tulajdonságainak beállítása

Importáljuk a `varos.txt` állomány adatait a *varos* táblába!
Az importálást követően nézzük meg a táblák tartalmát! Fontos, hogy a szöveges adatok ékezethelyesek legyenek, elférjenek a megfelelő mezőben, a terület értékeinél pedig a tizedesjegyek szerepeljenek.

A *városok* adatbázis tartalmát ritkán bővítjük, hiszen csak akkor kell rekordot hozzáadnunk, ha újabb települést nyilvánítanak várossá. (Nézzünk utána, mik a várossá nyilvá-

nítás feltételei!) Ennek ellenére sem változtathatatlan adathalmazról beszélünk, hiszen a lélekszám nem állandó. A valósan használt adatbázisról az adatok frissítését követően vagy rendszeres időközönként készítsünk biztonsági mentést! Az Access program esetén ez az adatbázist tartalmazó fájlról készített másolat, a legtöbb adatbázis-kezelő rendszer esetén pedig az adatbázis exportja. Az exportálás eredménye egy olyan szöveges fájl, amely azokat a parancsokat tartalmazza, amelyekkel az adatbázis szerkezete és adatai később helyreállíthatók. A `varosok_export.sql` fájlban található meg a *városok* adatbázis MySQL-ben készült változatának mentése. Ezt a fájlt akkor is érdemes végignéznünk és megbeszélnünk, ha nem foglalkozunk mélyebben az adatbázis-kezeléssel.

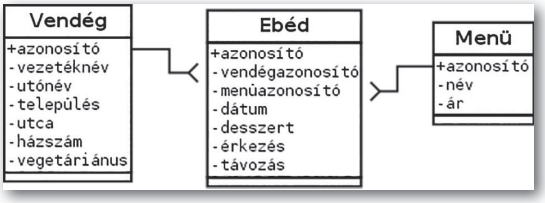
Importálás másképp

6. példa: Munka az *étkezés* adatbázissal

Az *étkezés* adatbázisban egy étterem menüjére előfizető vendégek adatait tároljuk. Nyilvántartják a számlázáshoz szükséges adatokat (név, cím), valamint azt, hogy a menü vegetáriánus-e. Mindennap négyféle menüből választhatnak a vendégek, ezek ára állandó. Eltároljuk azt is, hogy a vendégek melyik nap melyik menüt választották, kértek-e aznap desszertet, mikor érkeztek és távoztak. (Csak a 2020. év októberének adatait tartalmazza a hozzáférhető adathalmaz.)

Az adatbázis feltöltéséhez szükséges adatokat a `vendeg.txt`, `menu.txt`, `ebed.txt`, UTF-8 kódolású, tabulátorral tagolt szövegfájlok tárolják, amelyek első sora nem tartalmazza a mezőneveket.

Ehhez a képen látható adatbázis-szerkezetet használjuk.



► Az *étkezés* adatbázis szerkezete

A táblákat nem szükséges előre elkészítenünk, létrehozhatjuk az importálás során is. Ehhez a *Rekordok másolatának hozzáfűzése* helyett a *Forrásadatok importálása új táblába* pontot kell választanunk. A *Speciális* gombra kattintást követően a korábban megismertek mellett a mezőinformációs részben meg kell adnunk a mezőneveket. Az *Importálás varázsló* későbbi lépéseiben az azonosító (kulcs) beállítását szükséges még elvégeznünk. Ha már mindhárom táblát importáltuk, az idegen kulcsot a hivatkozási integritást megőrző kapcsolat megadásával állíthatjuk be.

Feladatok

Készítsük el a *javítás* adatbázist az alábbi leírásnak megfelelően! Az adatbázis feltöltéséhez szükséges adatokat az `ugyfel.txt`, `munka.txt`, `tipus.txt` nevű, UTF-8 kódolású, tabulátorral tagolt szövegfájlok tárolják, amelyek első sora tartalmazza a mezőneveket.

a) A mezőnevek tartalmát felhasználva rajzoljuk meg a *javítás* adatbázis szerkezetét!
b) Értelmezzük az egyes mezők szerepét, határozzuk meg a típusukat!
c) Milyen típust választottunk az ügyfél *irszám* mezőjének? Indokoljuk a választ!
d) Importáljuk a három szövegfájl tartalmát az adatbázisba! (A varázsló második lépésében ne felejtsük el megjelölni, hogy az első sor tartalmazza a mezőneveket!)
e) Állítsuk be az idegen kulcsokat a kapcsolatok létrehozásával! (A mezőnevek segítik a kapcsolatok meghatározását.)

Információ kinyerése az adattáblákból

Táblázatkezelés során általában egyetlen munkalap egy téglalap alakú tartományából nyertünk ki adatokat. Ennek során rendezhettük az adatokat, kereshettünk a munkalapon, vagy éppen a szűrésnek valamelyik változatát is segítségül hívhattuk.

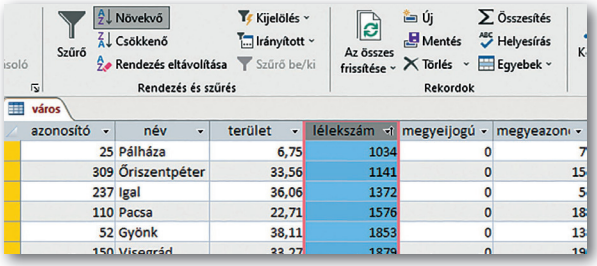
7. példa: Válaszok más szoftverekben is használt módszerekkel

Idézzük fel, hogy táblázatkezelő segítségével milyen kérdésekre tudnánk válaszolni ezekkel a műveletekkel! Ugyanezeket a problémákat az adatbázis-kezelőben is megoldhatjuk.

A már korábban elkészített, de a források között is megtalálható *városok* adatbázist használva adjuk meg, hogy

- melyik a legkisebb lélekszámú város;
- mekkora Komló területe;
- mely települések indulhatnak azon a pályázaton, amelyet 10 és 30 ezer fő közötti lélekszámú, legalább 250 km^2 területű városoknak írtak ki!

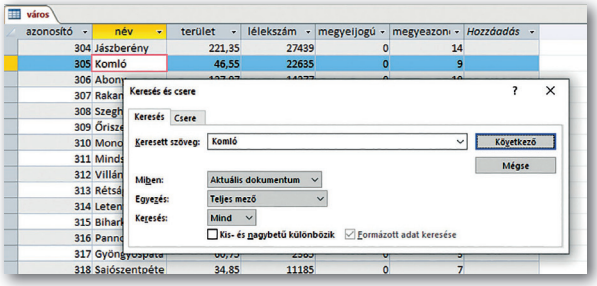
Rendezés



azonosító	név	terület	lélekszám	megyeijogú	megyeazoni
25	Pálháza	6,75	1034	0	7
309	Őriszentpéter	33,56	1141	0	15
237	Igal	36,06	1372	0	5
110	Pacsa	22,71	1576	0	18
52	Gyönk	38,11	1853	0	13
150	Visegrád	33,27	1879	0	19

A legkisebb lélekszámú várost a lélekszám oszlopára beállított növekvő rendezéssel lehet a legegyszerűbben meghatározni. A megjelenítés sorrendjét mindig a legutolsó rendezés szabja meg. Rendezés nélküli táblában az elsődleges kulcs a meghatározó.

Keresés



A keresés szóról szinte minden számítógép-használónak a `Ctrl + F` billentyűkombináció jut eszébe. A megnyitott táblában itt is használhatjuk.

Példaként keressük meg Komló adatait!

A keresés során figyeljünk a *hatókör* és az *egyezés* megfelelő beállítására! Próbáljuk megtalálni az „aba”, majd a „91” összes előfordulását a teljes mezővel egyezésben, majd bármely részében! Teljes egyezésnél mindkét esetben csak egy rekordot találunk, a mező bármely részét figyelve többet. Tehát az ilyen keresés nem más, mint a megadott karaktersorozat keresése – akár egy szám részeként.

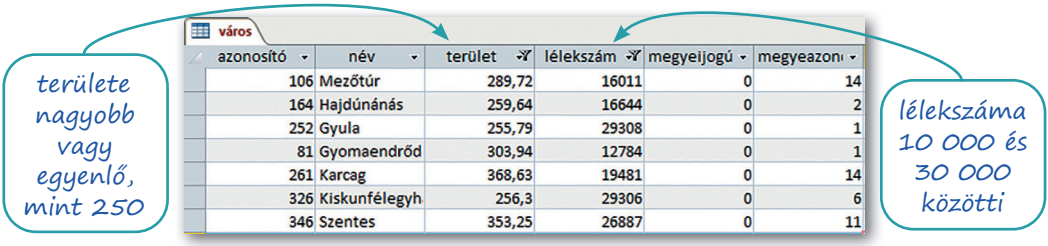
Keresés során a beírt szöveget tartalmazó rekordokon végig tudunk lépdelni, de az összes ilyen adatsort nem látjuk egyszerre. A keresésnél csak egy értéket adhatunk meg. Ha több feltételt is meg akarunk adni, ahhoz a szűrés funkcióra van szükségünk.

Szűrés

A szűréssel elérhetjük, hogy egyetlen táblának csak azokat a sorait jelenítse meg a program, amelyek az általunk meghatározott feltételnek megfelelnek. A táblázatkezelő helyben szűréséhez hasonlóan csak **ÉS** műveletet tartalmazó logikai kifejezést alakíthatunk ki szűrési feltételként.

Egy pályázaton csak azok a városok indulhatnak, amelyek lélekszáma 10 és 30 ezer fő között van, területük pedig legalább 250 km^2. Szűrjük ki ezeket a városokat!

Ehhez meg kell nyitni a táblát, és a megfelelő mezőnévre kattintva, a számszűrőknél beállítani a kívánt feltételt. A 10 és 30 ezer fő közötti lélekszámot megadhatjuk két részletben vagy intervallumszűréssel is. Érdemes megjegyezni, hogy az *„a és b érték között”* az adatbázis-kezelés során olyan értékekre teljesül, amelyekre igaz, hogy ≥*a* és ≤*b,* tehát a matematikában használt [*a;b*] zárt intervallumnak felel meg.



► Szűrési feltétel összeállítása

Hiába végzünk el egy szűrést, a táblát bezárva, majd ismét megnyitva az összes rekord megjelenik. Tehát a szűrés – a kereséshez hasonlóan – csak gyors tájékozódásra szolgál, a kapott adathalmazt áttekinthetjük vagy lemásolhatjuk, de – például újabb rekordok rögzítését követően – nem tudjuk ismét alkalmazni.

8. példa: Szövegből logikai kifejezés

Oldjuk meg a következő feladatokat rendezéssel, kereséssel vagy szűréssel!

1. Melyik a legkisebb területű város?
2. Melyik város neve az utolsó, és melyik az első az ábécérendben?
3. Mely városok területe nem haladja meg a 10 km^2-t? Hány ilyen város van?
4. Mely megyei jogú városok lélekszáma legfeljebb 100 ezer fő?
5. Mely 20 ezer főnél népesebb városoknak kisebb a területük 100 km^2-nél?
6. Mely 8 ezer főnél kevesebb lakosú városok területe 25 és 50 km^2 közötti? Hány ilyen város van?
7. Mely városokra nem teljesül, hogy népességük 10 ezer és 50 ezer fő között van?

Elég könnyű feladatokról van szó. Sokan úgy gondolják, hogy sikerült mindet megoldani. Nézzük meg alaposabban! Írjuk fel a feltételekben megfogalmazott logikai kifejezéseket a matematikában és a programozásban megszokott formában!

területe nem haladja meg a 10 km^2-t	terület ≤ 10
megyei jogú ÉS lélekszáma legfeljebb 10 ezer	megyei jogú = Igaz ÉS lélekszám ≤ 10000
100 km^2-nél kisebb ÉS 20 ezer főnél népesebb	terület < 100 ÉS lélekszám > 20000

Ha leellenőrizzük az utolsó sorban foglaltakat a szoftverben, akkor azt látjuk, hogy bizony ott szerepel az egyenlőség is. Ezen adatoknál nem tűnik fel a hiba, de érezzük, hogy adott esetben komoly gond lehet abból, ha csupán egy település is elesik egy pályázattól amiatt, mert egy egyenlőségjel lemaradt.

Szükségünk van egy eszközre, amelynek használata során a feltételt teljes pontossággal meg tudjuk fogalmazni. Ezt az eszközt lekérdezésnek nevezzük.

Lekérdezés

A lekérdezés megoldja az imént használt műveletekkel kapcsolatban felismert problémákat, tehát

- megváltozott tartalmú táblára is tudjuk alkalmazni,
- a szűrési kifejezés pontosan megadható,
- a szűrési kifejezésben nem csak az **ÉS** logikai művelet használható.

Lekérdezéssel a rendezési, keresési, szűrési feladatok megoldhatók. Az eddig megismert eszközöket természetesen továbbra is használhatjuk, de legyünk tudatában korlátaiknak.

A későbbiekben látni fogjuk, hogy a lekérdezésben nem csak egy táblát használhatunk, illetve lekérdezés segítségével számításokat is végezhetünk.

A lekérdezés legegyszerűbb formáját a szűréssel tekinthetjük egyenértékűnek: egy adattáblára vonatkozik, és a szűrési feltételnek megfelelő rekordok összes mezőjének értékét megjeleníti.

Ha az előző kérdésekre akarunk választ kapni, akkor a

```
SELECT *
FROM város
WHERE [feltétel]
```

formát kell használnunk. Ebben az alakban a félkövér karakterrel írtak mindig szerepelnek, az utolsó elemet kell helyettesítenünk a keresési feltétellel. A `SELECT *` azt jelenti, hogy a rekord minden mezőjét meg kell jeleníteni, a `FROM város` pedig megadja, hogy a *város* táblával dolgozunk.

9. példa: Feltételek megfogalmazása a lekérdezésekben

Nézzük meg, mit írjunk az állandó rész mögé a 8. példában szereplő, sorszámozott feladatok esetén:

```
3.… terület <= 10
4.… megyeijogú = Igaz AND lélekszám <= 100000
5.… lélekszám > 20000 AND terület < 100
6.… lélekszám < 8000 AND terület >= 25 AND terület <= 50
7.… NOT(lélekszám>= 10000 AND lélekszám <= 50000)
```

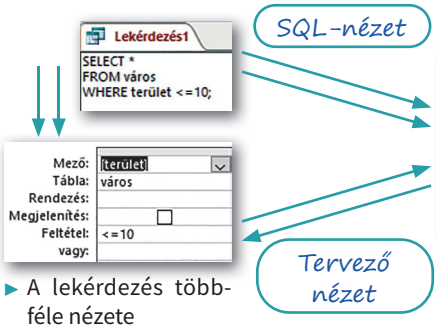
vagy – ha végiggondoljuk (akár egy számegyenes segítségével ábrázoljuk):

```
    … lélekszám < 10000 OR lélekszám > 50000
```

Ha mindez világos, akkor nézzük meg, miképpen használhatjuk az Access programban! Természetesen ugyanabban az adatbázisban dolgozunk, mint ahol a szűréseket végeztük. A *Létrehozás* > *Lekérdezéstervező* eszközt használva megnyílik egy *Lekérdezés1* nevű objek-

tum. A megjelenő párbeszédablakot most még hagyjuk figyelmen kívül, és a *Lekérdezés1* helyi menüjéből válasszuk az *SQL-nézet* pontot!

A párbeszédablakba írjuk be 3. feladat feltételt is tartalmazó parancsát, majd kattintsunk a *Lekérdezéstervezés* > *Futtatás* elemére!

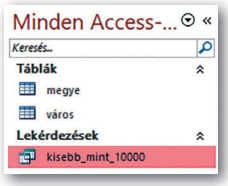


▸ A lekérdezés többféle nézete

▸ Adatlap nézet

A kapott lista egyezik a szűrés eredményével. Az Access programban a lekérdezésnek ezt az állapotát *Adatlap nézetnek* nevezzük. A lekérdezések helyi menüjében szabadon váltogathatunk a különböző nézetek között. Válasszuk ki a *Tervező nézetet!*

Tervező nézetben a lekérdezés ablakának alsó – táblázatos – részén, amelyet **lekérdezőrácsnak** hívunk, a mező sorában az a név látszik, amelyre a feltételt megfogalmaztuk, a feltétel sorában szerepel a relációs jel és a viszonyítási érték. Ez a két elem együttesen adja meg a korábban felírt feltételt.



A lekérdezést elmenthetjük, ha a nevére jobb gombbal kattintunk. Az elmentett lekérdezések az adatbázis ablakának bal oldalán megjelennek, nevükre duplán kattintva bármikor végrehajthatjuk azokat. A végrehajtással egyenértékű a *helyi menü* > *Megnyitás* lehetősége. A lekérdezés módosításához pedig a *helyi menü* > *Tervező nézet* pontját kell használnunk.

Bár minden lekérdezést megfogalmazhatunk SQL-nyelven, a későbbiekben elsősorban a lekérdezőrácson – *Tervező nézetben* – készítjük el a lekérdezéseket.

Feladatok

Próbáljuk megoldani az alábbi feladatokat lekérdezés készítésével és annak használatát mellőzve is! Nem kell megijednünk, ha csak az egyik módon járunk sikerrel, előfordul, hogy nem lehetséges, vagy még nincs minden szükséges ismeret a birtokunkban.

Használjuk az *étkezés* adatbázis tábláit!

a) Jelenítsük meg a vendégeket ábécérendben!
b) Adjuk meg az alsóvárosi lakosokat!
c) Beszélgettünk egy Ádám utónevű felsővárosi férfival. Ki lehetett ő?
d) Listázzuk ki azokat, akik valamilyen „út”-on laknak!
e) Mikor érkezett az első vendég 2020. 10. 07-én?
f) Mikor távozott az utolsó vendég 2020. 10. 07-én?
g) Hány vendég volt 2020. 10. 07-én?
h) Melyik menüt választotta Varga Tamara, amikor első ízben ott ebédelt?

Logikai műveletek a lekérdezésekben

Az előző leckében láttuk, hogyan tudjuk a szűrésnél megfogalmazott feltételeket a lekérdezésekben használni. Most már nem foglalkozunk a szűréssel, a problémákat lekérdezéssel oldjuk meg a *városok* adatbázisban.

10. példa: Relációs jelek a lekérdezésben

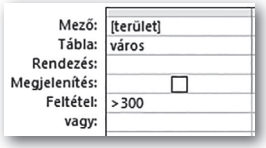
Az új ismeretek alapján könnyen tudunk lekérdezést készíteni az alábbi kérdésekhez. Az elkészült lekérdezéseket mentsük is el a zárójelben megadott néven!

- Mely városok területe nagyobb 300 km^2-nél? *(terület)*
- Mely városok lélekszáma haladja meg a 100 ezer főt? *(lélekszám)*

Íme a megoldások SQL-parancsként és lekérdezőrácson:

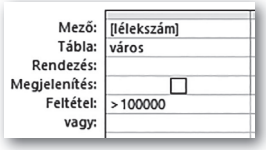
terület:

```
SELECT *
FROM város
WHERE terület > 300;
```



lélekszám:

```
SELECT *
FROM város
WHERE lélekszám > 100000;
```



Az ÉS művelet

11. példa: Az ÉS művelet többféle megfogalmazásban

Nézzük meg azt a feladatot, amelyben az előző két feltétel egyaránt szerepel!

Melyek a 300 km^2-nél nagyobb területű, ugyanakkor 100 ezer főnél népesebb városok? *(összetett1)*

A korábban megoldott szűrési feladatok szövegében a feltételeket az **és,** valamint a **vagy** szavak kapcsolták. Ezeket a szavakat, pontosabban az SQL-ben használt megfelelőjüket, az **AND** és **OR** logikai műveleteket szolgai módon be lehetett írni a lekérdezésbe. A feladat szövegében ezúttal ezt a szerepet az **ugyanakkor** szó tölti be, ilyen logikai művelet azonban nincs.

Meg kell szoknunk, hogy a mindennapokban megfogalmazott problémákhoz nem lehet tükörfordítással feltételt megadnunk. A bennük szereplő hétköznapi szófordulatok, megfogalmazások tartalmát meg kell értenünk, majd átfogalmaznunk úgy, hogy az elemi feltételeket az **ÉS, VAGY, NEM** szavak kapcsolják össze. Ezek az SQL-nyelvben használt logikai műveletek is, ezért az átfogalmazással általában a lekérdezésben használt feltételt is megalkottuk. Jelen esetben az **ugyanakkor** szó egyidejűséget fejez ki, az első feltétel a másodikkal egyszerre teljesül, ezért az **és** szóra cserélhetjük. Ha gondolkodunk egy keveset, akkor a feladatot másképp is meg tudjuk fogalmazni anélkül, hogy jelentésén változtatnánk.

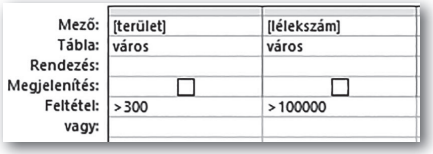
- Melyek a 300 km^2-nél nagyobb területű, valamint 100 ezer főnél népesebb városok?
- Melyek azok a városok, amelyek 300 km^2-nél nagyobbak, ráadásul 100 ezer főnél népesebbek is?

Keressünk még néhány, ezzel tartalmilag egyező megfogalmazást!

Gépeljük be az SQL-parancsot, és nézzük meg a *Tervező nézetet,* majd értelmezzük a látottakat!

összetett1:

```
SELECT *
FROM város
WHERE terület > 300
      AND lélekszám > 100000;
```



Lényegében annyi történt, hogy a *terület* és *népesség* lekérdezésekhez tartozó rácsokat összeillesztettük. A lekérdezőrács első sorában azok a mezőnevek szerepelnek, amelyekre feltételt fogalmazunk meg, alattuk pedig ugyanabban a sorban a relációs jelek és az értékek.

Ha két vagy több feltételt az ÉS művelettel kapcsolunk össze, akkor a lekérdezőrácson ugyanabba a sorba kerülnek.

Vizsgáljuk meg a *terület, lélekszám* és az *összetett1* lekérdezések eredményhalmazát! A *terület* és a *lélekszám* listák közös elemei alkotják az *összetett1* listáját, tehát ez a másik kettő metszete. Az adatbázis-kezelés során visszagondolhatunk a matematikából tanult halmazokra. Az adattábla teljes tartalmát tekintjük az alaphalmaznak, a feltételek pedig ennek részhalmazait határozzák meg. Egy-egy összetett problémához tartozó eredményhalmazt úgy is elképzelhetünk, mint a feltételek adta részhalmazokon végzett halmazműveletek eredményét. Nem kell minden esetben ilyen „hivatalosan” fogalmaznunk, de érdemes tudnunk, hogy a matematikából szerzett tudásunkat itt is alkalmazhatjuk.

A VAGY művelet

12. példa: A VAGY művelet alkalmazása

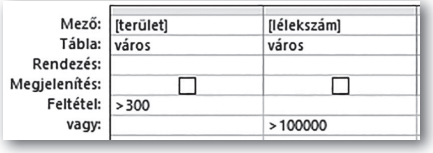
Nézzünk egy másik összetett feladatot!

Jelenítsük meg közös listában azokat a városokat, amelyek 300 km^2-nél nagyobb területűek, és azokat is, amelyek 100 ezer főnél nagyobb népességűek! *(összetett2)*

A fenti mondatban olvasott *és* szó ne tévesszen meg senkit, úgy kell tekintenünk, hogy vannak a 300 km^2-nél nagyobb területű városok, és ettől függetlenül vannak azok, amelyek 100 ezer főnél nagyobb népességűek. Nem elég csak az egyik halmaz városait megjeleníteni, hozzá kell venni még azokat is, amelyek a másik halmazban vannak. Nyilván azoknak nem kell duplán megjelenniük, amelyek mindkettőben szerepelnek. Ez pontosan megfelel a két halmaz uniójának. Az unió azon elemeket tartalmazza, amelyek legalább az egyik halmazban szerepelnek, tehát az egyik vagy a másik feltétel igaz rájuk, vagyis a két feltételt a **VAGY** művelettel, az SQL-parancsban pedig az **OR**-ral kapcsoljuk össze.

összetett2:

```
SELECT *
FROM város
WHERE terület > 300
      OR lélekszám > 100000;
```



A lekérdezés a lekérdezőrácson alig valamiben tér el az előzőtől, de figyelmesen megnézve a képet, észrevehetjük, hogy a feltételt megadó két érték nem ugyanabban a sorban van.

Ha két feltétel között **VAGY** kapcsolat van, akkor azokat a lekérdezőrácson külön sorba kell írnunk.

13. példa: A megfogalmazás egyértelműsége

Listázzuk ki azokat a városokat, amelyek területe meghaladja a 100 km²-t, és megyei jogúak, vagy lélekszámuk nagyobb 40 ezer főnél! *(összetett3)*

Egyesek úgy érezhetik, hogy a mondat megfogalmazása nem egyértelmű. Nem tudják eldönteni, hogy a területre vonatkozó feltétel csak a megyei jogú városokra vonatkozik-e, vagy a 40 ezer főnél népesebbekre is. Ez inkább a hétköznapi nyelven, szóban megfogalmazott feladatoknál fordulhat elő.

Az alábbi ábrák a lekérdezőrácson mutatják be a kétféle értelmezést. Határozzuk meg, hogy melyik melyik értelmezéshez tartozik!

Mező:	terület	megyeijogú	lélekszám
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:	>100	Igaz	
vagy:			>40000

► A feladat egyik értelmezése

Mező:	terület	megyeijogú	lélekszám
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:	>100	Igaz	
vagy:	>100		>40000

► A feladat másik értelmezése

Azt kell figyelnünk, hogy mit írtunk egy sorba, és mit külön. Ha ezeknek a lekérdezéseknek megnézzük az SQL-nézetbeli alakját, láthatjuk, hogy az Access híven követi azt, hogy amit egy sorba írunk, azok közé **AND** kerül, a külön sorokat pedig az **OR** határolja. (A parancssoros leírásból a felesleges zárójelek jelentős részét eltávolítottuk.)

```
összetett3a:
SELECT terület, megyeijogú, lélekszám
FROM város
WHERE (terület>100) AND (megyeijogú=True* OR lélekszám>40000);
összetett3b:
SELECT terület, megyeijogú, lélekszám
FROM város
WHERE (terület>100 AND megyeijogú=True)
  OR (terület>100 AND lélekszám>40000);
```

*A lekérdezőrácson feltétlenül ki kell írnunk az Igaz értéket, SQL-nézetben a True elhagyható, mivel a mező értéke önmagában is az igaz/hamis logikai érték.

Az első esetben a területre vonatkozó feltétel csak a megyei jogú városokra vonatkozik. A másik esetben a megyei jogúakra és a 40 ezer főnél népesebb városokra is igaz, hogy nagyobbak 100 km²-nél.

Ha megnézzük az összetett lekérdezések futtatásának eredményét, akkor azt látjuk, hogy az első kettőnél minden mező értéke megjelent, az utóbbiaknál pedig csak azok, amelyekre feltételt fogalmaztunk meg. A lekérdezőrácsra felvehetjük azokat a mezőket is,

amelyekre nem vonatkozik feltétel. A mezők láthatóságát a lekérdezőrács *Megjelenítés* sorában található jelölőnégyzetekkel szabályozhatjuk.

14. példa: A „között” alkalmazása

Adjuk meg az 50 és 100 ezer fő közötti népességű megyei jogú városokat! *(között)*

Vannak, akik az 50 ezer és 100 ezer közötti lélekszámú városok közé sorolják azt is, amelyik éppen 50 ezer lakosú, míg mások nem. Meg kell állapodnunk, hogy a továbbiakban a „között” kifejezést hallva az egyenlőséget megengedjük az intervallum mindkét végén.

Mező:	név	lélekszám	lélekszám	megyeijogú
Tábla:	város	város	város	város
Rendezés:				
Megjelenítés:	☑	☑	☐	☑
Feltétel:		>=50000	<=100000	Igaz
vagy:				

▶ Összeállított lekérdezés (A *lélekszám* mező kétszer is szerepel. Mivel ugyanazt a tartalmat jelenítik meg, ezért az egyiknél a megjelenítést kikapcsoltuk.)

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		>=50000 And <=100000	Igaz
vagy:			

▶ Átalakított lekérdezés

Megoldásunk az első ábrán látható. Ha elmentjük a megoldást, majd *Tervező nézetben* megnyitjuk, nem ugyanezt kapjuk vissza, hanem a második ábrán látható tartalmat. Érdemes megjegyezni, hogy a lélekszámra vonatkozó feltételt – amely egy intervallumot ad meg – ebben az egyszerűbb formában is bejegyezhetjük, tömörebbé, áttekinthetőbbé téve megoldásunkat. Ha valaki biztosan nem akar hibázni, használhatja az ábrán látható megoldást is.

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		Between 50000 And 100000	Igaz
vagy:			

▶ Az intervallumra szűrés új megoldása

Tagadás

15. példa: A tagadás használata

Határozzuk meg azokat a megyei jogú városokat, amelyek népessége nem éri el az 50 ezer főt, vagy meghaladja a 100 ezret! *(szélsőséges)*

- A három képen látható megoldás egyenértékű. Aki megjegyezte, hogy a VAGY kapcsolatot a feltételek külön sorba írásával fejezzük ki, az első képen látható formát használja. Nem szabad megfeledkezni azonban arról, hogy a megyei jogú tulajdonságot mindkét sorban fel kell tüntetni, hiszen az mindkét létszámkorlát esetén érvényes!

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		<50000	Igaz
vagy:		>100000	Igaz

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		<50000 Or >100000	Igaz
vagy:			

Mező:	név	lélekszám	megyeijogú
Tábla:	város	város	város
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:		Not (>=50000 And <=100000)	Igaz
vagy:			

- A második megoldáshoz az előző feladat második megoldása adja az ötletet. Ez nemcsak tömörebb leírást tesz lehetővé, hanem kevesebb hibalehetőséget rejt magában.
- A harmadik megoldást azok alkalmazzák, akik felismerik, hogy a lélekszámra vonatkozó feltétel tagadása az előző feladatban megfogalmazott kívánalomnak.

Jegyezzük meg: egy feltételt legegyszerűbben mindig az elé írt **NOT** szóval tagadhatunk.

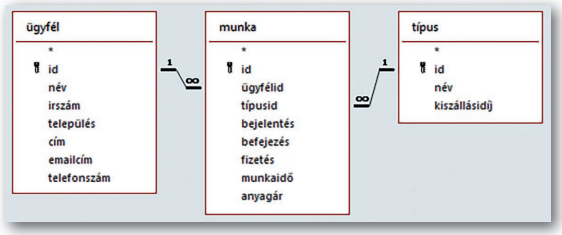
Feladatok

Oldjuk meg a következő feladatokat a *városok* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg a 100 km^2-nél kisebb területű megyei jogú városokat! *(f5a)*
b) Listázzuk ki a 150 és 250 km^2 közötti területű nem megyei jogú városokat! *(f5b)*
c) Határozzuk meg, mely városok azok, amelyek területe 150 és 250 km^2 között van, és népessége 50 ezer főnél kevesebb! *(f5c)*
d) Adjuk meg azokat a városokat, amelyek területe nem 150 és 250 km^2 közötti, de népességük 50 ezer főnél nagyobb! *(f5d)*
e) Soroljuk fel azokat a városokat, amelyek 300 km^2-nél nagyobb területűek, vagy népességük meghaladja a 100 ezer főt! *(f5e)*
f) Soroljuk fel azokat a megyei jogú városokat, amelyek 300 km^2-nél nagyobb területűek, vagy népességük meghaladja a 100 ezer főt! *(f5f)*

A szöveg mezőtípus

Az eddig készített lekérdezéseinkben a számokra vonatkozó feltételek mellett adott szöveges értékkel egyezést vizsgáltunk. Egy-egy probléma megoldásához ennél többre lehet szükségünk. Használnunk kell a kisebb/nagyobb relációt szövegek esetén is, szükséges a szövegrész előállítása, és gyakori igény a név szerinti sorrend felállítása.



► A *javítás* adatbázis szerkezete

Ebben a leckében a példák a *javítás* adatbázishoz tartoznak. Az alábbi kép az adatbázis szerkezetét adja meg. (A szerkezet bemutatására az Access program *Adatbáziseszközök* > *Kapcsolatok* pontján látható ábrát használjuk.)

16. példa: A szöveggel való egyezés

Az adatbázisban az ügyfélként feltüntetett személyek a lakásukban előforduló különböző típusú hibákat javíttatják meg. A javítás folyamatához tartozó adatokat a *munka* nevű kapcsolótáblában rögzítették, így a hibabejelentést, a munka befejezésének és a számla kifizetésének dátumát, a ráfordított munkaórák számát és a felhasznált anyagok árát.

A tanultak alapján az alábbi feladatokat meg tudjuk oldani lekérdezés segítségével:

- Adjuk meg a felsővárosi ügyfelek nevét! *(felsőváros)*
- Határozzuk meg azokat az ügyfeleket, akik nem Felsővárosban laknak! *(nemfelsőváros)*

Az első feladat megoldása nagyon könnyű volt, csak a település nevét kellett beírnunk. A begépelt szöveget a program automatikusan kiegészítette az ábrán is látható idézőjelekkel. *SQL-nézetben* feltétlen be kell gépelnünk az idézőjeleket, a lekérdezőrácson csak akkor szükséges, ha a beírt szöveg egyezik egy mező nevével, különben az Access mezőnévként értelmezi.

A második feladatot sokan a **NOT** logikai művelet segítségével oldják meg, mert a szöveg alapján ez tűnik magától értetődőnek. Átváltva *Adatlap* nézetbe, majd vissza *Tervező nézetbe,* azt tapasztaljuk, hogy a program átalakította a feltételt, a **NOT** helyére a <>, azaz a nem egyenlő jel került.

Mező:	név	település
Tábla:	ügyfél	ügyfél
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		"Felsőváros"
vagy:		

► A felsővárosiakat listázó lekérdezés

Mező:	név	település
Tábla:	ügyfél	ügyfél
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		Not "Felsőváros"
vagy:		

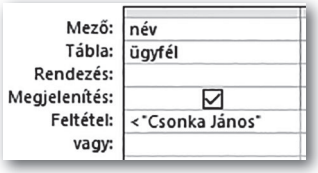
► A felsővárosi lakhelyet tagadó lekérdezés

Mező:	név	település
Tábla:	ügyfél	ügyfél
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		<>"Felsőváros"
vagy:		

► Felsővárossal nem egyező településnév

17. példa: A szöveg típussal kapcsolatos feltételek

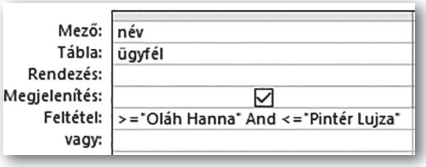
- A javítást végző cég ünnepi üdvözletet küld az ügyfeleinek. A leveleket ábécérendben küldik ki. Kiknek küldték el az üdvözletet Csonka János előtt? *(CsonkaJános)*
- Az ügyintézők között felosztották az ügyfeleket, ábécérendben. Istvánnál Oláh Hanna az első, Pintér Lujza pedig az utolsó. Kik tartoznak Istvánhoz? *(OláhPintér)*
- Évára bízták a számlák ellenőrzését az Oláh és a Pintér közötti vezetéknevű ügyfelek esetén. Adjuk meg az Évához tartozók névsorát! *(OP)*

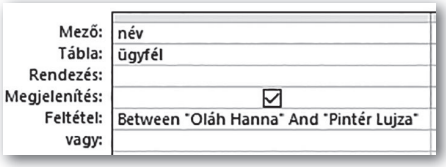


A kisebb és nagyobb reláció értelmezett a szöveg típus esetén is: az a név „kisebb”, amelyik az ábécében előrébb áll. Így könnyű dolgunk van, csak a megfelelő relációs jelet kell a név elé írnunk. Nézzük meg, kik előzik meg Csonka Jánost! Meglepő lehet, hogy Czirok Péter is közöttük van. A programozásban azt láttuk, hogy stringeknél az első eltérő karakter alapján dől el, melyik a kisebb, de a helyesírási szabályzat szerint az első eltérő betű betűrendben elfoglalt helye dönt. Jelen esetben a C betű előrébb van a Cs betűnél, ezért a Czirok is a Csonkánál.

Feladat

Olvassuk el, hogy mik a betűrendbe állítás szabályai (https://helyesiras.mta.hu/helyesiras/default/akh12#F2_4)!

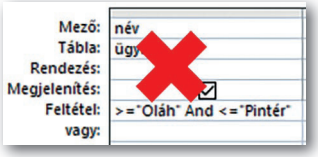




Mivel Oláh Hanna és Pintér Lujza is az eredményhalmaz része, ezért a ≥ és a ≤ relációs jeleket kell használnunk. A nevek rendezett sorozatában felfoghatjuk úgy is, mint egy zárt intervallumot, ezért a BETWEEN operátort is használhatjuk a feltétel megadásakor.

Az *OláhPintér* lekérdezés SQL-nyelven:

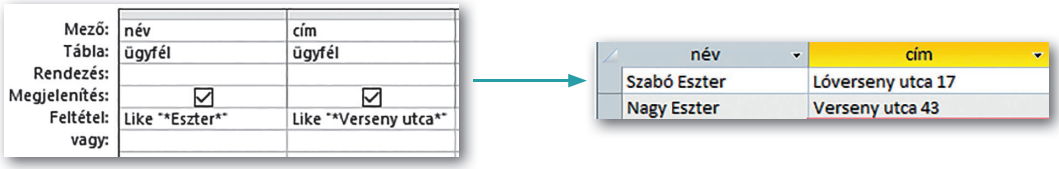
```
SELECT név
FROM ügyfél
WHERE név BETWEEN ”Oláh Anna” AND ”Pintér Lujza”;
```



Az Oláh és Pintér vezetéknevűekre vonatkozó feladat látszólag egyezik az előzővel – bár kevésbé pontosan adja meg a feltételt. Ha a megoldást a teljes nevet tartalmazóhoz hasonlóan állítjuk elő, akkor láthatóan nem az elvárt eredményhalmazt kapjuk meg, hiszen Pintér vezetéknevű egyáltalán nem szerepel a listában. Ha az ábrán látható feltételt nem a teljes névre, hanem csupán a vezetéknévre fogalmaznánk meg, a helyes eredmény állna elő. Ez a feladat így – bár megoldható az adatbázis-kezelő programmal – túlmutat a tankönyv tárgyalási szintjén.

18. példa: A mintaillesztés

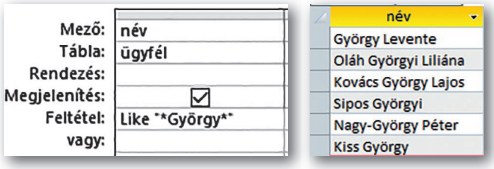
- A telefonos ügyfélszolgálatos helyét néhány percre egy másik munkatárs vette át. A legutolsó hívásból csak annyira emlékezett, hogy Eszter volt a Verseny utcából. Ki lehetett ő? *(Eszter)*
- A cég e-mailben köszönti az ügyfeleit a névnapjukon. Listázzuk ki a György utónevű ügyfelek nevét, e-mail-címét! *(György)*
- A cégnél rendszeresen ellenőrzik az ügyfelek adatait. Most azok vannak soron, akiknek a vezetékneve C-vel kezdődik. Adjuk meg az érintett ügyfelek nevét! *(Cbetű)*

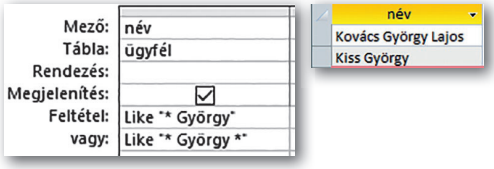


Ha belepillantunk az *ügyfél* táblába, azt látjuk, hogy az utónevet a vezetéknévvel együtt tároljuk a név mezőben, és az utca megnevezése sem önmagában áll. Ha lehetséges, hogy a keresett karaktersorozat csak része az adott mező értékének, akkor a nem ismert részek helyére a * karaktert írjuk. Ha úgy gondoljuk, hogy a keresett karaktersor bárhol állhat a mezőn belül, akkor elé és mögé is beírjuk a * karaktert. Ha beírjuk a feltételhez a helyettesítő karaktert tartalmazó szöveget, az Access program a feltételt automatikusan kiegészíti a **LIKE** operátorral. A fenti ábrát megnézve a lekérdezés nem az elvárt eredményt adja, mivel a „Verseny utca” a „Lóverseny utcára” is illeszkedik. Ezzel egyúttal azt is megtapasztaltuk, hogy a program szövegre vonatkozó feltételek vizsgálatakor nem különbözteti meg a kis- és nagybetűket.

Ha tudjuk, hogy a karaktersorozat a mező (kifejezés) elején helyezkedik el, akkor csak mögé kell írnunk a * karaktert. Az eredményhalmazt megvizsgálva azt hihetjük, hogy immár helyes megoldást adtunk. A lista tartalmilag megegyezik az elvárttal, de a következő feladat rávilágít arra, hogy átgondoltabbnak kell lennünk.

Az előző feladatban olvasott figyelmeztetés miatt érezzük, hogy a megoldás a György utónevűek megkeresésére a `Like "*György*"` feltétellel nem lesz alkalmas, mégis tanulságos ezzel kezdeni a próbálkozást. Számítottunk ugyan felesleges értékekre, de ilyen sokra nem. A sok felesleges sort többféle hiba okozza. Hiba, hogy megkaptuk azokat a neveket, amelyeknél a György vezetéknévként a név mező elején található. Szintén hiba, hogy megkaptuk a Györgyi nevűeket is.





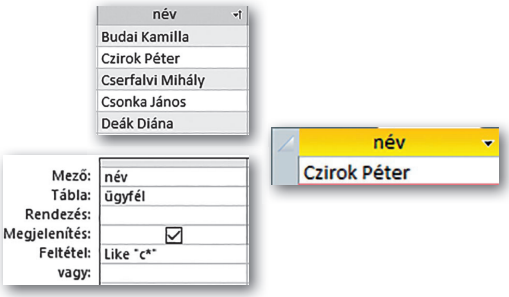
Nézzük, hogy a kívánalomnak megfelelő rekordokban miképpen fordul elő a György karaktersorozat! Lehet a végén, mint utolsó utónév. Ekkor szóköz áll előtte, és azelőtt bármi szerepelhet. Lehet, hogy egy közbülső utónév, ekkor viszont előtte és utána is szóköz áll, és az előtt és azt követően is bármely szöveg szerepelhet.

A *György* lekérdezés SQL-nyelven:

```
SELECT név
FROM ügyfél
WHERE név LIKE "* György" OR név LIKE "* György *";
```

Visszatérve az első feladatra, kijelenthetjük, hogy a helyes kimenet ellenére bizony hibás megoldást adtunk másodjára is. A szöveg nem szólt arról, hogy az Eszter vezeték- vagy utónév-e. Ha mindkettőt meg kell engednünk, akkor az ábrákon látható megoldás a helyes.

Ha a névre vonatkozó egyes eseteket soronként jegyezzük be, akkor vigyázzunk arra, hogy az utcára vonatkozó feltétel minden sorban szerepeljen! Az egysoros megfogalmazás nemcsak tömörebb, de kevesebb hibalehetőséget is rejt.



A három feladatból a C-vel kezdődő vezetéknevűek kigyűjtése a legegyszerűbb. Nem a megoldás nehézsége, hanem az eredményhalmazból levont tanulság a fontos. Az Access programban nem karakterekből, hanem betűkből álló szöveges kifejezésekkel dolgozunk. Ez a megoldási elvet nem befolyásolja, de ha meglepő eredményt kapunk, gondoljunk arra, hogy a magyar nyelvben két-, sőt háromjegyű betűk is vannak.

A **LIKE** operátor segítségével egy szöveg típusú kifejezést – általában egy mező értékét – összevethetünk egy mintával, amelyben a * tetszőleges számú, a ? pedig csak egyetlen karakter helyettesítésére szolgál. (Az Access programban a karakter szó helyett helyesebb betűt használnunk. Az SQL-nyelv szabványnak megfelelő változatai a * helyett a % jelet, a ? helyett pedig az _ jelet használják.)

Feladatok

Oldjuk meg a feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg azokat, akiknek az irányítószáma 3000 és 4000 között van! *(f6a)*
b) Listázzuk ki azokat, akiknek .hu-ra végződik az e-mail-címük! *(f6b)*
c) Soroljuk fel azokat az ügyfeleket, akiknek az utóneve *B*-vel kezdődik! *(f6c)*
d) Határozzuk meg azokat az ügyfeleket, akiknek a nevében van *a* és *e* karakter is! *(f6d)*
e) Adjuk meg azokat az ügyfeleket, akiknek a nevében nincs *a* és *e* karakter! *(f6e)*
f) Határozzuk meg azokat, akiknek az e-mail-címében van pont karakter a @ karakter előtt! *(f6f)*
g) Adjuk meg azokat, akiknek legalább két utónevük van! *(f6g)*
h) Listázzuk ki azokat, akiknek csak egy utónevük van! *(f6h)*

A dátum és az idő típus

Igen egyszerű dolgunk van, ha az adattábla szöveg típusú mezőjét töltjük fel értékekkel. Ha a mező beállításait nem módosítottuk, pontosan a begépelt szöveg jelenik meg mindenkinek a képernyőjén. Egész számoknál ugyanezt tapasztaljuk. Valós számnál már az operációs rendszer területi beállításaitól függően tizedesjelként pontot vagy vesszőt mutat, és így is kell begépelnünk. A dátum megjelenítése és bevitele sokféleképpen történhet. Az Access programban akkor tudjuk megszokott módon kezelni a dátumokat, ha az operációs rendszerben az éééé.hh.nn. formátum van beállítva. Ellenőrizzük ezt a beállítást, és szükség esetén módosítsuk! (Ellenőrizhető a *Vezérlőpult* > *Régió* > *További beállítások* > *Dátum* > *Dátumformátumok* > *Rövid dátum* pontjában. Magyar nyelvű Windows 10 esetén az éééé. HH.nn. a helyes beállítás.)

Tanultuk, hogy a táblázatkezelő a dátumokat egész számként, a napon belüli időt pedig törtként tárolja. Ez az adatbázis-kezelőben is így történik. Táblázatkezelőben 1900.01.01. a kezdő dátum, amely az 1 értéket kapja. Adatbázis-kezelőben beírhatjuk a mohácsi vész vagy Szent István koronázásának dátumát is, jól fogja megjeleníteni. A 0 értéknek az 1899.12.31. felel meg, a korábbi dátumokat negatív értékként kezeli. A kezelés csak technikai értelemben helyes, tudnunk kell, hogy a naptárreformokat nem veszi figyelembe.

19. példa: A dátummal kapcsolatos feltételekről

A következő feladatokat az *étkezés* adatbázis *ebéd* tábláját használva oldjuk meg!

- Adjuk meg a 2020. 10. 06-án fogyasztott ebédek adatait! *(október6)*
- Határozzuk meg, mely napokon érkezett vendég 11:40 előtt! *(korán)*
- Listázzuk ki, mely napokon nem volt 11:40 előtt érkező vendég! *(nemvolt)*

Mező:	azonosító	személyazonosító	menüazonosító	dátum	desszert	érkezés	távozás
Tábla:	ebéd	ebéd	ebéd	ebéd	ebéd	ebéd	ebéd
Rendezés:							
Megjelenítés:	☑	☑	☑	☑	☑	☑	☑
Feltétel:				#2020.10.06.#			
vagy:							

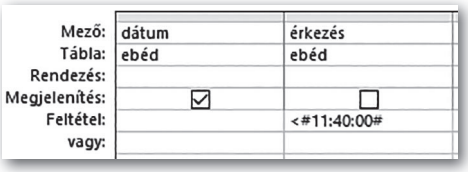
► Az *október6* lekérdezés a mezők egyenkénti feltüntetésével

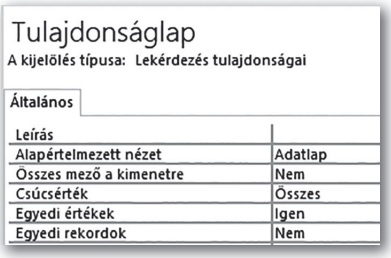
Mező:	ebéd.*	dátum
Tábla:	ebéd	ebéd
Rendezés:		
Megjelenítés:	☑	☐
Feltétel:		#2020.10.06.#
vagy:		

► Az *október6* lekérdezés a * karakter felhasználásával

Az *október6* lekérdezésnél az összes mező megjelenítését kétféleképpen is megoldhatjuk. Kiválaszthatjuk egyenként az összes mezőt, de használhatjuk a * karaktert is – jelen esetben az *ebéd*.* formában. Egy-egy feladat megoldása során értelmezés kérdése, hogy az azonosító mező része-e az ebéd jellemzőinek. Technikai szempontból mindenképpen, ezért nem követünk el hibát a megjelenítésével. Feltételként a 2020.10.06. értéket kell begépelnünk. Az Access program az adott mezőből továbblépve automatikusan kiegészíti a # karakterekből álló párral, jelezve, hogy dátum típusú adatról van szó. (A legtöbb adatbázis-kezelő esetén a dátumokat is idézőjelek közé írjuk.) Ha nem ez történik, akkor valószínűleg a dá-

tumot gépeltük el, vagy az adott mező nem dátum típusú. Ha az egyszerűbb megoldást választjuk, akkor a feltételt megadó *dátum* oszlop megjelenítését kapcsoljuk ki, ugyanis ez a * karakter miatt mindenképpen látható lesz.



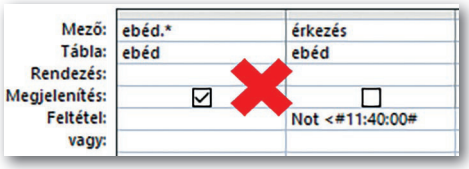


A *korán* lekérdezésnél a feltételhez beírt időpontot (például 11:40) a program kiegészíti óra:perc:másodperc alakra, és a # karakterrel jelzi, hogy idő típusú adatról van szó. A feladat nem kéri, hogy az érkezés időpontja is jelenjen meg, ezért kapcsoljuk ki a megjelenítését! Az eredményhalmazt megvizsgálva azt látjuk, hogy egyes napok többször is megjelennek. A többszörös megjelenés jelen esetben számunka semmilyen hasznos információt nem hordoz, ezért kapcsoljuk ki! Erre a *Lekérdezéstervezés* > *Megjelenítés, elrejtés* > *Tulajdonságlap* > *Egyedi értékek* ponton van lehetőség, az *Igen* érték beállításával.

A *korán* lekérdezés SQL-ben:

```
SELECT DISTINCT dátum
FROM ebéd
WHERE érkezés<#11:40:0#;
```

A DISTINCT kulcsszó az eredményhalmaz ismétlődéseit kiszűri, minden sor pontosan egyszer jelenik meg.

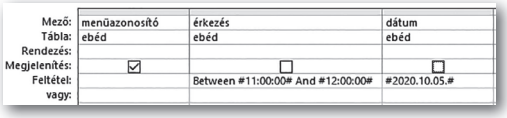


Sokan azt gondolják, hogy a *nemvolt* lekérdezést megkapjuk, ha a *korán* lekérdezés feltételét tagadjuk. A lekérdezést végrehajtva azt látjuk, hogy sok olyan nap megjelenik, amely az előző listának is része volt. Vegyük észre, hogy ezzel a feltétellel azokat a napokat adjuk meg, ahol volt 11:40-kor vagy azt követően érkező vendég. Ezeken a napokon érkezhettek néhányan korábban is. Az adatbázis-kezelés sokak számára azt jelenti, hogy megadjuk a rögzített értékek egy részhalmazát, ebben a kérdésben viszont éppen olyan értékeket várunk, amelyeket nem rögzítettek. Ez a feladat nehéz, jelenlegi tudásunkkal még nem tudjuk megoldani, később is csak arra tudunk válaszolni, hogy azon napok közül, amelyeken volt vendég, mely napokon nem volt 11:40 előtt érkező.

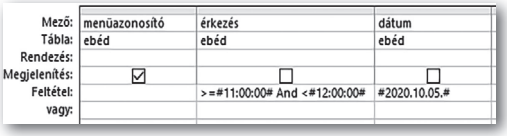
20. példa: Az időintervallumok kezelése

- Listázzuk ki, hogy milyen menüket szolgáltak ki 11:00 és 12:00 között 2020. 10. 05-én! Mindegyik menü betűjele pontosan egyszer jelenjen meg! *(11és12között)*
- Adjuk meg, hogy milyen menüket szolgáltak ki a nap 12. órájában 2020. 10. 05-én! Mindegyik menü betűjele pontosan egyszer jelenjen meg! *(óra12)*

A *11és12között* lekérdezést elkészítve azt látjuk, hogy az adott intervallumon belül több A, B és C menüt adtak ki. Ezek többszöri felsorolása felesleges, a *korán* lekérdezésnél látott módon, az egyedi értékek beállításával az ismétlődő megjelenést elkerülhetjük.

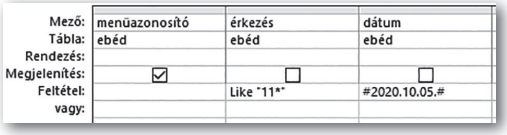


Mező:	menüazonosító	érkezés	dátum
Tábla:	ebéd	ebéd	ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		Between #11:00:00# And #12:00:00#	#2020.10.05.#
vagy:			



Mező:	menüazonosító	érkezés	dátum
Tábla:	ebéd	ebéd	ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		>=#11:00:00# And <#12:00:00#	#2020.10.05.#
vagy:			

Az *óra12* lekérdezésnél érdemes tisztázni, hogy a nap első órája 0 óra 0 perc 0 másodperckor kezdődik, a 01:00:00 pedig már a második óra kezdete. Tehát a nap 12. órájához a 11:00:00 hozzátartozik, de a 12:00:00 már nem. A képen látható megoldásban 11 óránál szerepel az egyenlőségjel, 12 óránál már nem!



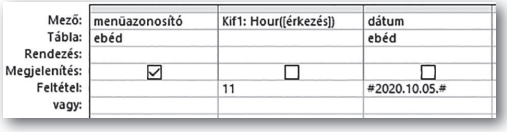
Mező:	menüazonosító	érkezés	dátum
Tábla:	ebéd	ebéd	ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		Like "11*"	#2020.10.05.#
vagy:			

Gondoljuk végig, hogy mely időpontok tartoznak ide! Mindegyik a 11-es számmal kezdődik. Ha az előző anyagrésznél figyeltünk, akkor megkísérelhetjük a mintaillesztést az *érkezés* mezőre, azaz egy idő típusú adatra alkalmazni. 11 a kezdete a mezőnek, mögötte bármi szerepelhet. Az Access program a LIKE operátor használatakor automatikusan szöveggé konvertálja a dátum és idő típusú adatokat, ezért tudjuk használni a mintaillesztést. Időpontoknál jól gondoljuk meg a használatát, mert – ha a második óra lett volna a kérdés – az 1* feltétel illeszkedett volna az 1:15:00-s és a 11:15:00-s értékre is. Az ilyen problémákat elkerülhetjük, ha az elválasztó karaktereket is beírjuk a keresőkifejezésbe.

Dátum- és időfüggvények

A mintaillesztéshez vezető ötletet más módon is felhasználhatjuk. A 11 nem pusztán két karakter, amely a mezőben előre került, hanem az időpont óráját jelenti. Valójában az *érkezés* mező órájára kellene feltételül szabni, hogy az értéke 11 legyen. Táblázatkezelés kapcsán tanultunk olyan függvényeket, amelyek az időpont óráját, percét, másodpercét – `Óra()`, `Percek()`, `Mperc()` – határozzák meg. Ilyen szerepű függvények az adatbázis-kezelő programban is léteznek: **Hour()**, **Minute()**, **Second()**. E függvények paramétere dátum/idő típusú érték.

Az *óra12* feladatnak az ábrán látható lekérdezés is helyes megoldása. Ha végiggondoljuk, hogy milyen problémákat lehet az időpontokra vonatkozóan megfogalmazni, érezzük, hogy a relációs jelek segítségével megadott feltételek a legáltalánosabban használhatók. Természetesen a másik két megoldási módot is érdemes ismerni.



Mező:	menüazonosító	Kif1: Hour([érkezés])	dátum
Tábla:	ebéd		ebéd
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		11	#2020.10.05.#
vagy:			

Az *Óra12* lekérdezés SQL-nyelvű megfelelője:

```
SELECT DISTINCT menüazonosító
FROM ebéd
WHERE Hour(érkezés)=11 AND dátum=#10/5/2020#;
```

Az SQL-nyelvű leírásban a dátum megadására csodálkozhatunk rá. A # határolók mellett a hónap, nap, év sorrend és a / karakter mint határoló jelenthet újdonságot.

Ahogy az időadatok óráját, percét, másodpercét meg tudjuk határozni megfelelő függvények segítségével, úgy a dátumokat is felbonthatjuk év, hónap, nap számértékekre. Ehhez a táblázatkezelőben használt Év(), Hónap(), Nap() függvényekkel egyezően működő **Year()**, **Month()**, **Day()** függvényekre van szükségünk.

Feladatok

Oldjuk meg a következő feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Listázzuk ki a 2017. augusztus 15. és szeptember 15. között bejelentett munkák adatait! *(f7a)*
b) Határozzuk meg a 2017 decemberében kezdődő tél hónapjaiban befejezett munkák adatait! A megoldáshoz használjuk a BETWEEN operátort! *(f7b)*
c) Kérdezzük le a 2017 novemberében bejelentett munkákat mintaillesztés segítségével! *(f7c)*
d) Kérdezzük le a 2017 júniusában kifizetett munkákat dátumfüggvények segítségével! *(f7d)*
e) Adjuk meg a korábbi évek valamelyikében, a feladatmegoldás napján kifizetett munkákat! A megoldáshoz használjunk függvényt! *(f7d)*

Rendezés

Az eddig megismert lekérdezésekben elsősorban a szűrésre koncentráltunk, az eredmény megjelenési sorrendje közömbös volt számunkra. A valóságban gyakran nem elégszünk meg ennyivel, igenis tudni akarjuk, hogy az elérhető buszok közül melyik ér oda a leghamarabb, melyik szállodai szoba a legolcsóbb, vagy ki nyerte a versenyt. Ahhoz, hogy a kérdésekre válaszolni tudjunk, a rendezéssel kell megismerkednünk.

21. példa: Az egy- és a többkulcsú rendezés

Az alábbi feladatok az *étkezés* adatbázisra vonatkoznak.

- Listázzuk ki a vendégek adatait vezetéknevük szerint rendezve! *(vezetéknév)*
- Adjuk meg a vendégek adatait vezetéknév, azon belül utónév szerint rendezve! *(név)*
- Listázzuk ki a Nagyfaluban lakók nevét, utcáját és házszámát utca, azon belül házszám szerint rendezve! *(cím)*
- Adjuk meg az összes vendég vezeték- és utónevét, települését – ebben a sorrendben feltüntetve az adatokat – település, azon belül név szerint rendezetten! *(összetett)*

Mező:	vendég.*	vezetéknév	keresztnév
Tábla:	vendég	vendég	vendég
Rendezés:		Növekvő	Növekvő
Megjelenítés:	☑	☐	☐
Feltétel:			
vagy:			

54	Balog	Noel	Felsőváros	Verseny utca	57
32	Balogh	Adél	Alsóváros	Pellérdi út	21
146	Balogh	Eszter	Felsőváros	Görgey Artúr u	4
35	Balogh	Hunor	Nagyfalu	Stiglicfogdosó	32
83	Balogh	Levente	Nagyfalu	Kút utca	3
53	Barna	Panna	Alsóváros	Mátyás Flórián	23

▶ Teljes név szerint rendezve

Korábban tanultuk, hogyan lehet a *vezetéknév* feladatot lekérdezés nélkül megoldani, de mindenképpen ez az első lépés a rendezés tanulása során. Beállítása rendkívül egyszerű, a lekérdezőrács *Rendezés* sorában kell meghatározni a rendezés irányát.

Mező:	vendég.*	vezetéknév
Tábla:	vendég	vendég
Rendezés:		Növekvő
Megjelenítés:	☑	☐
Feltétel:		
vagy:		

51	Balog	Lilien	Kisfalu	József utca	17
146	Balogh	Eszter	Felsőváros	Görgey Artúr u	4
32	Balogh	Adél	Alsóváros	Pellérdi út	21
83	Balogh	Levente	Nagyfalu	Kút utca	3
35	Balogh	Hunor	Nagyfalu	Stiglicfogdosó	32
53	Barna	Panna	Alsóváros	Mátyás Flórián	23

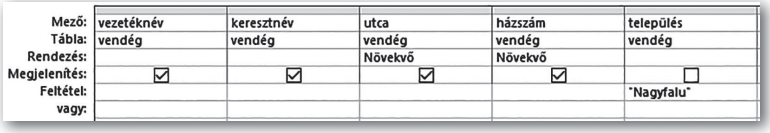
▶ Vezetéknév szerint rendezve

A *vezetéknév* lekérdezés SQL-nyelvű megfogalmazása:

```
SELECT *
FROM vendég
ORDER BY vezetéknév;
```

(Az **ORDER BY** vezetéknév **ASC**; forma lenne a teljes, de az ASC [ascending] elhagyható, mert a növekvő rendezés az alapértelmezett.)

A lekérdezés futtatását követően nézzük meg az eredményhalmaz rekordjainak sorrendjét! Azt találjuk, hogy a Balogh vezetéknevűek között a sorrend gyakorlatilag véletlenszerű. A *név* lekérdezés hétköznapi szóhasználattal azt kéri, hogy a *vezetéknév,* azon belül pedig az *utónév* döntsön a sorrendben. Ezt **többkulcsú rendezésnek** hívjuk. Ha két rekord első kulcs szerinti értéke egyezik, sorrendjüket a második kulcshoz tartozó érték fogja megszabni. A lekérdezőrácson a rendezésre kijelöltek közül az első kulcs oszlopának meg kell előznie a második kulcs oszlopát. A *név* lekérdezést futtatva azt látjuk, hogy a Balogh vezetéknevűek az utónevük szerinti sorrendben szerepelnek.



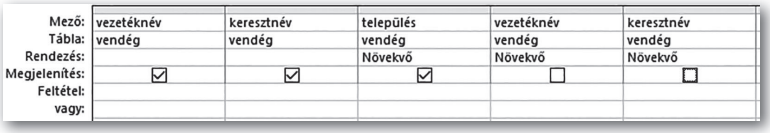
Mező:	vezetéknév	keresztnév	utca	házszám	település
Tábla:	vendég	vendég	vendég	vendég	vendég
Rendezés:			Növekvő	Növekvő	
Megjelenítés:	☑	☑	☑	☑	☐
Feltétel:					"Nagyfalu"
vagy:					

Horváth	Boglárka	Lahti utca	44
Lengyel	Zsófia	Mécses utca	46
Nagy	Eszter	Mécses utca	8
Tóth	Letícia	Örs utca	31

A *név* lekérdezés SQL-nyelvű megfogalmazása:

```
SELECT *
FROM vendég
ORDER BY vezetéknév, utónév;
```

A *cím* lekérdezés elkészítése semmivel sem nehezebb, mint a *név* lekérdezésé. A lényeget az eredményhalmaz sorrendjének vizsgálatánál látjuk. A Mécses utcában a 46-os házszám alatt lakó megelőzi a 8-as szám alattit. A sorrend nem az elvárásaink szerint alakul. Nézzük meg, hogy mi az oka! A tábla *Tervező nézetében* látjuk a szerkezetet és azt, hogy a házszám szöveg típusú mező. Idézzük fel a tanultakat! Két szöveg típusú érték sorrendjét az első eltérő karakter sorrendje dönti el. Ezért a szövegként tárolt 46 előrébb kerül a listában, mint a 8. Ennek a tulajdonságnak azért választottuk a szöveg típust, mert a *házszám* mezőben tároltuk a 2/B értéket is. Ahhoz, hogy a házszámokat szám típusú értékként tárolhassuk – a helyes rendezés érdekében –, az egyéb karaktereket le kell választani róla. Erre szolgálnak egy-egy űrlapon az *épület,* illetve *lépcsőház* mezők.



Mező:	vezetéknév	keresztnév	település	vezetéknév	keresztnév
Tábla:	vendég	vendég	vendég	vendég	vendég
Rendezés:			Növekvő	Növekvő	Növekvő
Megjelenítés:	☑	☑	☑	☐	☐
Feltétel:					
vagy:					

Ahogy a *cím* lekérdezésben, úgy sok más esetben sem tulajdonítunk jelentőséget a mezők megjelenési sorrendjének, az *összetett* lekérdezés viszont kifejezetten előírja azt. Az oszlopok megjelenítési sorrendje – *vezetéknév, utónév, település* – és a rendezési előírás sorrendje – *település, vezetéknév, utónév* – ellentmondóak. Ilyen esetben vegyük fel a rendezési előírásnak megfelelő sorrendben az érintett mezőket, állítsuk be a rendezést, de ne jelenítsük meg azokat, amelyek már előbb szerepeltek a megjelenítési sorrendben. A fenti ábrán egy lehetséges megoldás látszik. Ha valaki SQL-nyelven oldja meg a feladatot, nem kell ilyen „trükkhöz" folyamodnia:

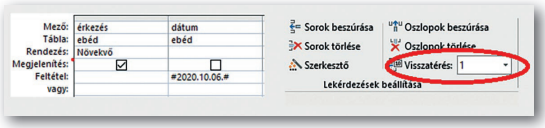
```
SELECT vezetéknév, utónév, település
FROM vendég
ORDER BY település, vezetéknév, utónév;
```

SQL-nyelven egyéb egyszerűsítési lehetőség is van, a rendezést megadhatjuk a megjelenített oszlop sorszámával is:

```
SELECT vezetéknév, utónév, település
FROM vendég
ORDER BY 3, 1, 2;
```

22. példa: A sorrendileg első rekordok megjelenítése

- Hánykor érkezett a legelső vendég 2020. 10. 06-án? *(legelső)*
- Hánykor távozott az utolsó vendég 2020. 10. 06-án? *(legutolsó)*
- Melyik nap kértek először desszertet? *(desszert)*
- Melyik volt az első öt olyan nap, amikor desszertet kértek? *(desszert5)*



A legelső kérdésre az előzőleg megoldott feladatok alapján könnyen tudnánk válaszolni. Növekvő sorrendbe állítjuk az érkezési időket, és leolvassuk az első értéket az *érkezések* oszlopából. A sok érték megjelenése azonban zavaró, hiszen nem az összes, hanem csak a legelső válaszol a kérdésre. A lekérdezőrácsban kialakított feltétellel együtt megjelölhetjük, hogy az előállt lista elejéről hány elemet jelenítsen meg. Ezúttal az 1 értéket írjuk a lekérdezés visszatérési tulajdonságához. Ez egyenértékű a lekérdezés *Tulajdonságlap* > *Csúcsérték* beállítással.

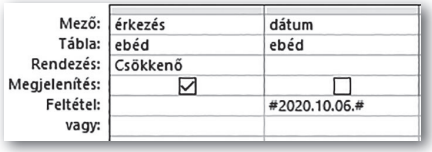
A *legelső* lekérdezés SQL-nyelven:

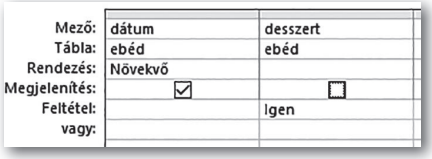
```
SELECT TOP 1 érkezés
FROM ebéd
WHERE dátum =#10/6/2020# ORDER BY érkezés;
```

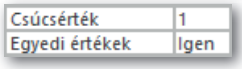
A legutolsó távozó vendég esetén megint a sorba rendezés a kulcs, csak a lista utolsó elemét kell választanunk. Mivel az adatbázis-kezelő programok csak az első néhány elem megjelenítésére képesek, ezért meg kell fordítanunk a rendezés irányát a megoldáshoz. A lekérdezéshez tartozó visszatérési értékek számát az ábrán nem jelöltük.

A *legutolsó* lekérdezés SQL-nyelven:

```
SELECT TOP 1 érkezés
FROM ebéd
WHERE dátum =#10/6/2020# ORDER BY érkezés DESC;
```







A *desszert* lekérdezés futtatása nem várt eredményt hoz. Hiába állítottuk a visszatérési értékek számát 1-re, vég nélkül sorolja a 2020.10.01. értéket. A desszertrendelés egyáltalán nem különleges, ezért már a legelső napon is sok vendég megtette. Az Access program nem tud választani az egyező értékek közül, így az összeset megjeleníti.

Ez nem jelenti azt, hogy a feladatot ne tudnánk megoldani jelenlegi tudásunkkal. Korábban megtanultuk kiszűrni az ismétlődéseket. A *Tulajdonságlapon* az egyedi értékek beállítása eltávolította az eredményhalmaz ismétlődő rekordjait, ha most ezt alkalmazzuk, az elvárt értéket, egyetlen dátumot kapunk. Elgondolkodhatunk azon, hogy vajon először az ismétlődéseket távolítja-e el, azután korlátozza a megjelenített sorok számát, vagy fordítva. Ezt a *desszert5* lekérdezést elkészítve magunk is megválaszolhatjuk.

A **TOP n** módosító segítségével a megjelenített rekordok számát az első *n* darabra korlátozzuk. A program nem rangsorolja a kulcskifejezés szerint egyező értékeket, így az *n*. rekorddal egyezőek mind megjelennek. (Az SQL-szabvány nem ismeri a TOP módosítót, abban a hasonló szerepű LIMIT található meg, amit a lekérdezés végén kell megadni.)

Feladatok

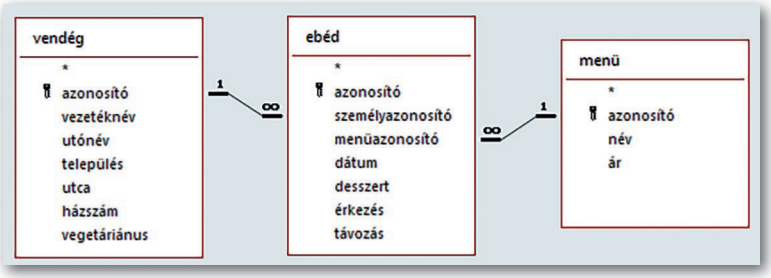
Oldjuk meg a feladatokat a *városok* adatbázist használva, lekérdezőrács segítségével!

a) Listázzuk ki a megyei jogú városokat betűrendben! *(f8a)*
b) Adjuk meg azon városok nevét és lélekszámát, amelyek nevében a város vagy a falu karaktersorozat szerepel! *(f8b)*
c) Határozzuk meg a három legnagyobb területű várost! *(f8c)*
d) Melyik a legkisebb népességű város azon nem megyei jogú városok közül, amelyek területe a 200 km^2-t meghaladja? *(f8d)*

Adatok több táblából

Az eddigi feladatok megoldásához elegendő volt egyetlen tábla adataival dolgozni. Sokszor olyan problémákat oldunk meg, amelyekhez egyszerre több tábla adatait kell használnunk.

Amikor a feladatmegoldás során használt táblákat létrehoztuk, meghatároztuk a közöttük lévő kapcsolatokat is. Ha felhasználjuk ezeket a kapcsolatokat, akkor az egymással összefüggésben álló táblákat tekinthetjük úgy is, mint egyetlen – olykor gigantikus méretű – **virtuális táblát.** Ez a tábla fizikailag nem létezik, de egy, több táblát használó lekérdezéssel el tudjuk készíteni, ahogy az alábbi ábrán látható is.



► Az *étkezés* adatbázis szerkezete

Mező:	vezetéknév	utónév	település	utca	házszám	vegetáriánus	dátum	desszert	érkezés	távozás	név	ár
Tábla:	vendég	vendég	vendég	vendég	vendég	vendég	ebéd	ebéd	ebéd	ebéd	menü	menü
Rendezés:	Növekvő	Növekvő										
Megjelenítés:	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Feltétel:												
vagy:												

vezetéknév	utónév	település	utca	házszám	vegetáriánu	dátum	desszert	érkezés	távozás	név	ár
Bakos	Gergely	Felsőváros	Rakéta utca	27	☐	2020.10.12.	☐	12:10:05	12:24:04	húsimádó	1410
Bakos	Gergely	Felsőváros	Rakéta utca	27	☐	2020.10.22.	☐	11:42:14	11:56:02	tésztás	1290
Bakos	Gergely	Felsőváros	Rakéta utca	27	☐	2020.10.20.	☐	12:34:21	12:47:12	tésztás	1290
Bakos	Olivér	Nagyfalu	Rezeda dűlő	39	☐	2020.10.15.	☐	13:28:34	13:41:31	diétás	1380
Bakos	Olivér	Nagyfalu	Rezeda dűlő	39	☐	2020.10.02.	☑	13:21:01	13:34:42	húsimádó	1410
Bakos	Olivér	Nagyfalu	Rezeda dűlő	39	☐	2020.10.20.	☐	13:35:09	13:48:06	diétás	1380

► Az *étkezés* adatbázis összetartozó adatai

Az ábrán látható, összetartozó adatokat előállító lekérdezés SQL-nyelven:

```
SELECT vendég.vezetéknév, vendég.utónév, vendég.település, vendég.utca,
    vendég.házszám, vendég.vegetáriánus,
    ebéd.dátum, ebéd.desszert, ebéd.érkezés, ebéd.távozás,
    menü.név, menü.ár
FROM vendég, ebéd, menü
WHERE vendég.azonosító=ebéd.vendégazonosító
    AND ebéd.menüazonosító=menü.azonosító
ORDER BY vendég.vezetéknév, vendég.utónév
```

(A tábla nevét csak akkor kötelező a mezőnév elé írni, ha a felhasznált táblákban az adott mezőnév több helyen is előfordul.)

Ha visszagondolunk a fejezet elején tanultakra, akkor eszünkbe juthat, hogy ez a táblázat a normalizálás előtti állapotot mutatja. Nagyon nagy a redundancia, hiszen a nevek, címek sokszor szerepelnek, és számtalan sorban olvasható a menü neve és ára is. Innen emeltük ki a vendégek és a menük adatait egy-egy külön táblába. Vegyük észre, hogy a táblának

pontosan annyi sora van, mint az *ebéd* táblának, és oszlopainak száma az egyes táblák oszlopszámának összege.

A lekérdezések elkészítéséhez nemcsak táblákat, hanem a lekérdezések eredményeként előálló virtuális táblákat is használhatunk. A későbbiekben ezt – a táblákat egyesítő – lekérdezést használjuk majd fel. Ezt megtehetjük, ha meg akarjuk tudni, hogy Bakos Gergely mely napokon milyen menüt választott, azonban felesleges, ha Bakos Olivér címére vagyunk kíváncsiak. Utóbbi kapcsán miért dolgoznánk több mint 4000 rekorddal, ha ehhez elegendő kevesebb mint 200 vendég adatait átnézni?

Tehát, ha egy táblát felhasználva nem tudunk válaszolni egy kérdésre, akkor használjuk fel pontosan csak azokat, amelyek a megoldáshoz szükségesek, de ne többet. Érdemes néhány példán keresztül megismerni, hogy mire kell figyelni a táblák kiválasztásánál.

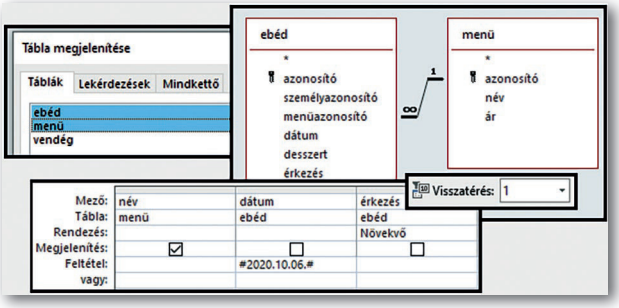
23. példa: **Több tábla használata**

- Határozzuk meg, hogy Bakos Gergely melyik nap milyen menüt választott! A lista legyen dátum szerint rendezett! *(Bakos)*
- Adjuk meg, hogy 2020. október 6-án melyik menüt szolgálták fel az elsőnek érkező vendégnek! *(első)*
- Listázzuk ki azokat a vegetáriánusokat, akik nem csak vegetáriánus menüt rendeltek! *(vegetáriánus)*

A *Bakos* lekérdezésben a négy szükséges mező három táblában található, ezért mindhármat használnunk kell. Amikor a lekérdezéskészítés első lépéseként kiválasztjuk a táblákat, a beemeltek között megjeleníti azokat a kapcsolatokat, amelyeket az adatbázis létrehozása során meghatároztunk. A feltétel beírását és a rendezés megadását követően készen is van a lekérdezés.

Mező:	dátum	név	vezetéknév	utónév
Tábla:	ebéd	menü	vendég	vendég
Rendezés:	Növekvő			
Megjelenítés:	☑	☑	☐	☐
Feltétel:			"Bakos"	"Gergely"
vagy:				

Az *első* lekérdezésben a menü *nevét* kell megjeleníteni, a *dátum* mezőre állítunk be feltételt, a sorrendet pedig az *érkezés* mező szabja meg. Az *ebéd* és a *menü* táblára van szükségünk, amelyek között kapcsolat van. Az eredményhalmaznak csak az első elemét kell megjeleníteni. Az ábra négy része ezeket a beállításokat mind mutatja. Ennek eredménye egyetlen szó, mely szerint diétás menüt szolgáltak fel.



A *vegetáriánus* lekérdezésben a neveket kell megjelenítenünk a *vendég* táblából, a feltétel pedig a *vendég* és a *menü* tábla mezőire vonatkozik. Látszólag csak két táblát igényel, a *vendég* és a *menü* táblákat. Ha ezt a kettőt vesszük fel, és a lekérdezőrács tartalmát az ábra szerint alakítjuk, akkor az eredményhalmaz 60 rekordot tartalmaz. Több mint 4000 ebédelésből nem olyan sok ez. Jelenítsük meg a menük nevét is, és rendezzük betűrendbe a neveket! Furcsának tűnik, hogy minden vegetáriánus vendéghez mindhárom nem vegetáriánus menü hozzátartozik. Mivel két olyan táblát használtunk, melyek között nem volt kapcsolat, ezért a 20 vegetáriánushoz mind a 3 lehetséges menüt hozzárendelte, ezért kaptunk eredményül 60 sort.

Mező:	vezetéknév	utónév	vegetáriánus	név
Tábla:	vendég	vendég	vendég	menü
Rendezés:				
Megjelenítés:	☑	☑	☐	☐
Feltétel:			Igaz	<>"vegetáriánus"
vagy:				

Vegyük fel az *ebéd* táblát is! A lekérdezőrácson nem módosítva 62 rekordot kapunk, de egyetlen esetben sem szerepel a húsimádó menü. Itt pontosan azok az ebédek szerepelnek, ahol a vegetáriánusok nem vegetáriánus menüt rendeltek.

Ahhoz, hogy pontosan a kérdésre válaszoljunk, állítsuk be az egyedi értékek megjelenítését a *Tulajdonságlapon!* Láthatjuk, hogy 20 személyből 19 fő élt ilyen választással.

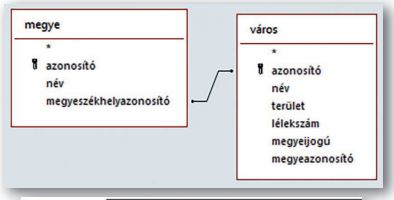
Ha egy adatbázis több táblából áll, a lekérdezést úgy építjük fel, hogy kiválasztjuk azokat a mezőket, amelyeket meg kell jelenítenünk, és azokat, amelyekre feltétel vonatkozik. A lekérdezésbe beemeljük azokat a táblákat, amelyekben szerepel kiválasztott mező. Ha az így kiválasztott táblák között nincs kapcsolat, akkor kiválasztjuk azokat a táblákat is, amelyek segítségével a kapcsolat megteremthető.

Egyedi kapcsolatok

Az eddigi lekérdezésekben azokat a táblák közötti kapcsolatokat használtuk, amelyeket az adatbázis készítése során határoztunk meg. Előfordulhatnak olyan esetek is, amikor nem erre vagy nem csak erre van szükség. A *városok* adatbázis kapcsán erre látunk példát.

24. példa: Előre nem megadott kapcsolatok használata

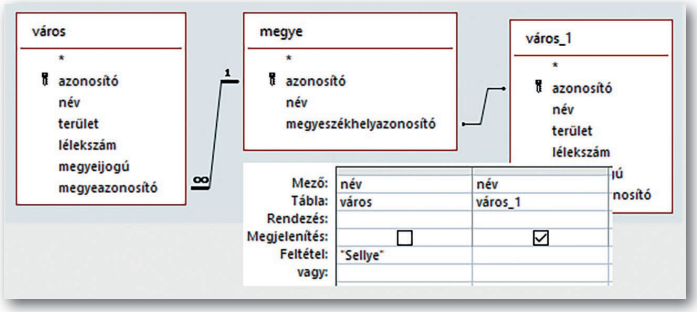
- Melyik város Heves megye székhelye? *(Heves)*
- Mi a székhelye annak a megyének, ahol Sellye található? *(Sellye)*



Mező:	név	név
Tábla:	megye	város
Rendezés:		
Megjelenítés:	☐	☑
Feltétel:	"Heves"	
vagy:		

A *Heves* lekérdezésben egy város nevére vagyunk kíváncsiak, és a megye nevét ismerjük. Természetesen mindkét táblát használnunk kell. A kérdést nem tudjuk megválaszolni az adatbázisban adott táblák közötti kapcsolattal, mert az a városhoz rendel megyét, nekünk pedig a megyéhez kell város. A *megye* táblában a *megyeszékhelyazonosító* tulajdonság idegenkulcs-szerepű, most az általa megadott kapcsolatra van szükségünk. A táblák választásakor automatikusan bekerülő kapcsolatot törölnünk kell, és az ábrán láthatót létrehozni.

Azt meg tudjuk mondani, hogy Sellye melyik megyében van. Ha ismerjük Sellye megyéjét, akkor – az előző lekérdezés alapján – képesek vagyunk meghatározni, hogy mi a megyeszékhelye. Aki gondolja, el is készítheti a megyét meghatározó lekérdezést, majd azt az ábra szerint a *város* tábla újbóli felvételével kiegészítve meg tudja alkotni a *Sellye* lekérdezést. Ebben a lekérdezésben sem csupán az adatbázisban megtalálható eredeti kapcsolatot használtuk, hanem egy újat is.



A *Sellye* lekérdezés SQL-nyelvű megoldása:

```
SELECT város_1.név
FROM város, megye, város AS város_1
WHERE város.megyeazonosító=megye.azonosító
    AND megye.megyeszékhelyazonosító=város_1.azonosító
    AND város.név="Sellye";
```

Feladatok

Oldjuk meg a feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Határozzuk meg, mikor dolgoztak a Pellérdi úton! *(f9a)*
b) Listázzuk ki, hogy ki, mikor és milyen típusú problémát javíttatott 2019-ben! *(f9b)*
c) Határozzuk meg, ki volt az első ügyfele a cégnek! *(f9c)*
d) Adjuk meg, milyen típusú munkához tartozott a legnagyobb értékű javítás! *(f9b)*

Számított értékek

Emlékezzünk vissza az adatbázis-tervezés bemutatása során használt, diákokról szóló típusmondatra és táblázatra! Ott arra jutottunk, hogy nem tároljuk el a diákok korát, hiszen az folyamatosan változik, helyette a születési évet (dátumot) rögzítjük, mert abból mindig ki tudjuk számítani a kort. Szinte alig van olyan adatbázis, amelyben nem tudunk számítást igénylő problémát kitűzni.

25. példa: Matematikai műveletek használata

A *városok* adatbázisban a népsűrűséget, az *étkezés* adatbázisban az adott napon az ebédért fizetett összeget, a *javítás* adatbázisban a munkadíjat kell kiszámítanunk. A számítások során sokszor csak a matematikából ismert műveleteket kell használnunk, de előfordulhat, hogy az itt tanult függvényeket is alkalmaznunk kell.

- Határozzuk meg a *városok* adatbázisban a Tolna megyei városok népsűrűségét! A városokat népsűrűség szerint csökkenően jelenítsük meg! *(Tolna)*
- Az *étkezés* adatbázis adatait használva listázzuk ki azok nevét, akik kevesebb mint 12 perc alatt végeztek az ebéddel! Írassuk ki az érkezési és távozási időt, majd értelmezzük az eredményt! *(kisebb12)*
- A *javítás* adatbázis adatai alapján határozzuk meg a 2018-as év számláinak végösszegét! Egy munkaóra díja 3500 Ft. A befizető neve, címe és a végösszeg jelenjen meg! *(fizetendő)*
- A *javítás* adatbázis adatai alapján adjuk meg, kik voltak azok, akik – bár befejezték náluk a munkát – csak a befejezést követő évben fizettek! Az ügyfél azonosítója és neve jelenjen meg! *(jövőre)*

város.név	Kif1
Szekszárd	330,234732031575
Dombóvár	229,599898063201
Nagymányok	206,647940074906
Bonyhád	177,945661214306
Tolna	155,522724074856
Paks	120,865784008307
Simontornya	113,479160508424
Bátaszék	95,5167531854648
Dunaföldvár	75,7763417698797
Tamási	70,6208128628852
Gyönk	48,6224088165836

A *Tolna* lekérdezés elkészítésénél a népsűrűség meghatározása jelenti az újdonságot. A népsűrűség nem más, mint két mező értékének hányadosa: *lélekszám/terület*. Ezt a kifejezést begépelhetjük, vagy a lekérdezőrácson a mező sorában a *helyi menü* > *Szerkesztés*… pontját választva ki is kattintgathatjuk. Utóbbi megoldásnak a hasonló bonyolultságú képleteknél csak annyi előnye van, hogy a gépelési hibáktól megvéd bennünket.

A lekérdezést futtatva az ábrán látható eredményt kapjuk. A képet alaposan megnézve néhány szokatlan dolog tűnik fel. A népsűrűség oszlopának fejében a *Kif1* szöveg olvasható, a népsűrűség értékénél indokolatlanul sok tizedesjegy van, valamint a városok oszlopa felett a *város.név* látható. Azt szoktuk meg, hogy a lekérdezések eredményét megnézve az eredménytáblázat első sorában a mezőneveket olvashatjuk. A népsűrűség nem egyetlen mezőt jelöl, hanem kettőnek a hányadosát. Ilyen esetben az Access program az oszlopot automatikusan elnevezi. A népsűrűség oszlopa a *Kif1* nevet kapta. Az első oszlopban a mező neve azért egészült ki a tábla nevével, mert a felhasznált táblák közül legalább

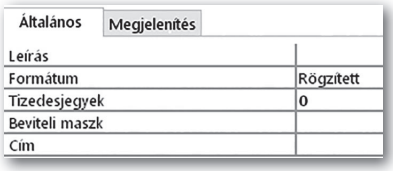
Mező:	város: név	népsűrűség: [lélekszám]/[terület]	név
Tábla:	város		megye
Rendezés:		Csökkenő	
Megjelenítés:	☑	☑	☐
Feltétel:			"Tolna"
vagy:			

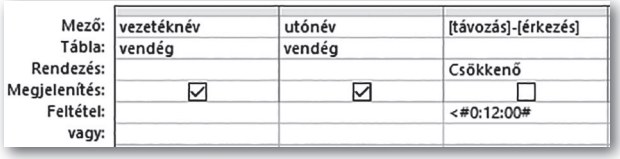
kettő tartalmazott *név* nevű mezőt, így a tábla neve tette egyértelművé, hogy melyikkel dolgozunk. Ha visszaváltunk a lekérdezőrácsra, akkor a *Kif1* szöveget ott is megtaláljuk: *Kif1*: [*lélekszám*]/[*terület*]. Egy oszlopot mi magunk is elnevezhetünk, ha a mező sorban a mezőnév vagy kifejezés elé beírjuk a kívánt feliratot, attól kettősponttal elválasztva.

A tizedesjegyek számának csökkentése is egyszerű feladat. A lekérdezőrácson ki kell választani az érintett mezőt, majd megjeleníteni a *Tulajdonságlapját*. A *Formátum* és a *Tizedesjegyek* sorában kell az ábra által megjelenített beállításokat megtennünk.

A *Tolna* lekérdezés SQL-nyelvű megoldása (a kerekítést az SQL-lekérdezésben kerekítő függvénnyel lehet előállítani, míg az Accessben készített megoldás formázással jelenítette meg):

```
SELECT város.név AS város, Round(lélekszám/terület, 0) AS népsűrűség
FROM város, megye
WHERE város.megyeazonosító=megye.azonosító
    AND megye.név="Tolna"
ORDER BY 2 DESC;
```





A *kisebb12* lekérdezésben két időértéket tartalmazó mező különbségét kell meghatározni. Gondoljunk vissza arra, amikor a dátum és idő típusú adatokkal ismerkedtünk! Valójában számértékekről van szó, csak a megjelenítési formátum az idő. A dátumot az egészrész adja, a napon belüli időpontot pedig a törtrész. Ellenőrzésképpen jelenítsük meg a különbség értékét is! Nem meglepő, hogy nemnegatív, egynél kisebb valós számokat látunk. Az előző lekérdezésnél látható módszerrel megpróbálhatjuk az időformátumra alakítást, de erre itt nincs mód a *Tulajdonságlap* segítségével. Jelenítsük meg az érkezés és a távozás időpontját is! Az a meglepő, hogy az érintett rekordok mindegyikében pontosan 12 percig tartó ebédeket látunk, tehát egyik sem teljesíti a feltételt. Nos, ez igaz, az ok a számítógép számábrázolásában keresendő. Mivel a 12 perc pontosan 120-ad része az 1 nap időtartamnak, ezért szorozzuk meg 120-szal a különbség értékét! Azt fogjuk látni, hogy nem 1 egészet kapunk eredményül, hanem annál egy nagyon kicsivel kisebb számot, az eltérést a 15. tizedesjegyben találjuk.

Természetesen nem várjuk el, hogy mindenki ilyen szintű alapossággal vizsgáljon meg minden feladatot, de fontos, hogy bemutassuk, bizony az elvileg helyes megoldással is kaphatunk hibás eredményt. Az idő óra, perc, másodperc részből áll. Ezek egészek. Ha ezeket használjuk, nem jelenik meg a valós számok ábrázolásából adódó probléma. Nézzük azt a feltételt, amely erre épül:

```
Hour([távozás])*3600+Minute([távozás])*60+Second([távozás]) -
    (Hour([érkezés])*3600+Minute([érkezés])*60+Second([érkezés])) < 720
```

Mező:	név	település	cím	Év: Year([befejezés])	fizetendő: [anyagár]+[kiszállásidíj]+3500*[munkaidő]
Tábla:	ügyfél	ügyfél	ügyfél		
Rendezés:					
Megjelenítés:	☑	☑	☑	☐	☑
Feltétel:				2018	

Ha ezt a feltételt használjuk, a helyes eredményt kapjuk, mert az eredménylista üres lesz.

A *fizetendő* lekérdezés azt mutatja, hogy a számításhoz szükséges mezőket lehet, hogy több táblából kell összegyűjtenünk. A *fizetendő* oszlopában szereplő számok pénzösszegek. Állítsuk be a mező megjelenítését tartalmának megfelelően, azaz jelenjen meg a pénznem is! Ezt a *Tolna* lekérdezés mintájára könnyen megoldhatjuk.

Mező:	id	név	Kif1: Year([befejezés])
Tábla:	ügyfél	ügyfél	
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			<Year([fizetés])
vagy:			

Mező:	id	név	eltérés: Year([fizetés])-Year([befejezés])
Tábla:	ügyfél	ügyfél	
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			1
vagy:			

A *jövőre* feladathoz a fenti ábrákon látható két lekérdezést készítettük. Futtatásuk eredménye ugyanaz a lista. Vajon mindkettő helyes?

Az első esetben a feltétel szerint a fizetés éve későbbi, mint a befejezés idejének éve. A második megoldás azt mondja, hogy a fizetés évének és a befejezés évének különbsége pontosan 1. Vegyük észre, hogy az első megoldás megadhat olyan ügyfelet is, aki elvégeztette a munkát 2019-ben, de már 2021 volt, amikor fizetett, tehát a lista elvileg tartalmazhat olyan adatsort is, amely nem felel meg a leírásnak. A helyes megoldáshoz egy számítás vezetett. Ettől eltérő megoldást is adhatunk, de azt nem tudjuk elkerülni, hogy számítást végezzünk.

Feladatok

Oldjuk meg a következő feladatokat lekérdezőrács segítségével!

a) A *javítás* adatbázis adatai alapján határozzuk meg, hogy a 2018 márciusában és áprilisában bejelentett munkák esetén a bejelentést követően hány nappal fejezték be a munkát! A befizető neve, címe és az eltelt napok száma jelenjen meg! *(f10a)*

b) A *javítás* adatbázis adatai alapján adjuk meg, mely munkáknál fordult elő, hogy az anyagköltség a duplája volt a kiszállási díjnak! *(f10b)*

c) Az *étkezés* adatbázis adatait használva listázzuk ki, hogy 2020. 10. 08-án kik ebédeltek a leghosszabb ideig! A három legtöbb időt eltöltő nevét adjuk meg! *(f10c)*

d) Az *étkezés* adatbázis adataiból dolgozva adjuk meg, hogy az egyes vendégek menynyit fizettek 2020. 10. 08-án! A desszert aznap 500 Ft-ba került. Ügyeljünk arra, hogy az érték helyes legyen a desszertet fogyasztó és nem fogyasztó vendégek esetén is! A vendég neve és a számla összege jelenjen meg! *(f10d)*

Aggregáló függvények

Az eddig szerzett tudásunkkal már sok problémát meg tudunk oldani, de több, a maga természetességében megfogalmazott feladattal nem boldogulunk. Azt bármely nap esetén meg tudjuk válaszolni, hogy mikor érkezett az első vendég, és mikor távozott a legutolsó, de ehhez két lekérdezésre van szükségünk. Hát még akkor milyen nehéz helyzetben lennénk, ha ezt az étterem minden munkanapjára meg kellene adnunk! Ez utóbbihoz először még a dátumokat is ki kellene gyűjtenünk, ráadásul egy újabb dátum bejegyzését nem követné a megoldás.

26. példa: Az aggregáló függvények használata

Az étkezés adatbázist használva oldjuk meg az alábbi feladatokat!

- Adjuk meg egyetlen lekérdezéssel, hogy 2020. 10. 08-án mikor érkezett az első vendég, és mikor távozott a legutolsó! *(tólig)*
- Listázzuk ki, hogy az egyes napokon mikor érkezett az első vendég, és mikor távozott a legutolsó! *(tólignaponta)*
- Határozzuk meg, hogy az egyes napokon hányan kértek desszertet! *(desszertnaponta)*
- Adjuk meg, hogy mely napokon kért legalább 100 fő desszertet! *(desszert100)*

Mező:	Nyitás: érkezés	Zárás: távozás	dátum
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Min	Max	Where
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			#2020.10.08.#
vagy:			

A *tólig* feladatot két lekérdezéssel meg tudjuk oldani, egyikben az *érkezés* szerint növekvően, a másikban a *távozás* szerint csökkenően rendezve a rekordokat. Akár azt is meg tudjuk adni, hogy melyik vendég volt az. E két érték megadását egyszerűbben is megoldhatjuk. Az egyik oszlop legkisebb és egy másik legnagyobb értékét határozzuk meg az adott dátumra szűrés mellett! A lekérdezőrácson jelenítsük meg az érkezés, távozás és dátum mezőket, majd kattintsunk a *Lekérdezéstervezés* > *Megjelenítés/elrejtés* > *Összesítés* gombra! Ennek hatására egy új, *Összesítés* nevű sor jelenik meg a rácson. Mivel az érkezés legkisebb értékére van szükségünk, az összesítés sorának lenyíló listájából válasszuk ki a **Min**, a távozásnál pedig a **Max** értékeket! A *dátum* oszlopában a **Where** elem szerepeljen. A futtatás eredménye egyetlen értékpár lesz, az október 8-i adatok.

A *tólig* feladat SQL-nyelvű megoldása:

```
SELECT Min(érkezés) AS nyitás, Max(távozás) AS zárás
FROM ebéd
WHERE dátum=#10/08/2020#;
```

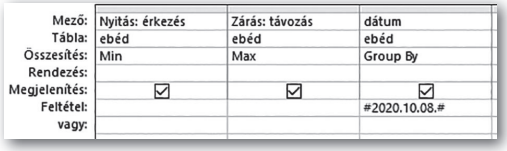
Mező:	Nyitás: érkezés	Zárás: távozás	dátum
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Min	Max	Group By
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:			
vagy:			

A *tólignaponta* feladattal látszólag nehéz megbirkózni, hiszen az előző megoldás kibővítéséhez mindenképpen ismernünk kellene a dátumok listáját. Készítsünk másolatot a *tólig* lekérdezésről, majd módosítsuk azt! Töröljük a dátumot a feltételből, majd az *Összesítés* sorában a **Where** szót cseréljük a **Group By** kifejezésre! A lekérdezést futtatva pontosan a kívánt listát kapjuk. A **Group By** választásakor az történik, hogy a program az adott oszlopban szereplő mező vagy kifejezés szerint

csoportokat képez az egyéb feltételeknek megfelelő rekordokból. Ezt követően a program minden ilyen csoportban meghatározza és megjeleníti a minimális és a maximális értéket. Figyeljünk arra, hogy ekkor a *dátum* oszlop megjelenítését be kell kapcsolnunk.

A *tólignaponta* feladat SQL-nyelvű megoldása:

```
SELECT Min(érkezés) AS nyitás, Max(távozás) AS zárás, dátum
FROM ebéd
GROUP BY dátum;
```

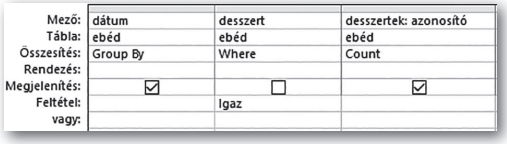


Mező:	Nyitás: érkezés	Zárás: távozás	dátum
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Min	Max	Group By
Rendezés:			
Megjelenítés:	☑	☑	☑
Feltétel:			#2020.10.08.#
vagy:			

Megnézve az így kapott listát, az egyik sorban megtaláljuk a választ a *tólig* feladat kérdésére. Mi lenne, ha csak ezt listáznánk ki? Készítsünk egy másolatot ezúttal a *tólignaponta* lekérdezésből *tóligmásképp* néven! Egészítsük ki ezt a lekérdezést a dátumra vonatkozó feltétellel! Lefuttatva a lekérdezést, a megjelenített rekord egyezik azzal, amit a *tólig*nál láttunk. Vajon eltér-e, és ha igen, miben a *tólig* és *tóligmásképp?* Amikor a dátum a **WHERE** alatt szerepel, a program kigyűjti a feltételnek megfelelő rekordokat, majd azokra alkalmazza a minimum és a maximum meghatározását. Ha a **GROUP BY** oszlopába írjuk a feltételt, akkor először – ebben a feladatban – elvégzi a csoportosítást, tehát minden dátumra megállapítja a minimum- és a maximumértékét, majd ebből az eredményhalmazból szűr. Ez a második megoldás erőforráspazarló, hiszen így minden dátumra el kell végeznie a számítást, míg az első esetben csak az adott dátumhoz tartozó rekordokra.

A *tóligmásképp* lekérdezés SQL-nyelven:

```
SELECT Min(érkezés) AS nyitás, Max(távozás) AS zárás
FROM ebéd
GROUP BY dátum
HAVING dátum=#10/08/2020#;
```

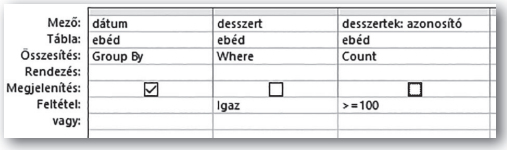


Mező:	dátum	desszert	desszertek: azonosító
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Group By	Where	Count
Rendezés:			
Megjelenítés:	☑	☐	☑
Feltétel:		Igaz	
vagy:			

A *desszertnaponta* feladat megoldásánál már tudjuk, hogy az összesítés bekapcsolása a lényeges pont. Természetesen itt is dátum szerint csoportosítunk, a desszertre vonatkozó feltételnél pedig tudatosan a WHERE-t használjuk. Gyakran elkövetik azt a hibát, hogy valamelyik feltételként felhasznált oszlopnál választják ki a számlálást (COUNT) az *Összesítés* sorban. Ilyenkor a feltétel nem a tábla adataira, hanem az eredménytáblára vonatkozik, ezért céljainktól eltérő eredményt kapunk. A COUNT használata során minden olyan rekordot figyelembe vesz, ahol a választott mező nem üres. Mivel a legtöbb mező minden rekordban ki van töltve, ezért látszólag bármelyiket választhatjuk. Biztosan nem hibázunk, ha olyan mezőt választunk, amely sohasem üres. Az azonosító szerepű mező mindig megfelel ennek a kritériumnak.

A *desszertnaponta* lekérdezés SQL-nyelven:

```
SELECT dátum, Count(azonosító)
FROM ebéd
WHERE desszert
GROUP BY dátum;
```



Mező:	dátum	desszert	desszertek: azonosító
Tábla:	ebéd	ebéd	ebéd
Összesítés:	Group By	Where	Count
Rendezés:			
Megjelenítés:	☑	☐	☐
Feltétel:		Igaz	>=100
vagy:			

A *desszert100* feladat megoldása nem más, mint a *desszertnaponta* módosítása azzal, hogy a darabszámra vonatkozó feltételt megfogalmazzuk. Lényeges, hogy erre itt nem a WHERE-t kell használnunk, mivel a feltétel a számlálás eredményére vonatkozik.

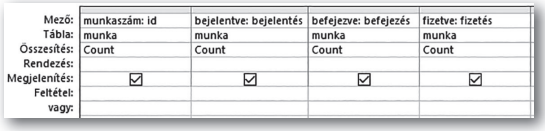
A *desszert100* lekérdezés SQL-nyelven:

```
SELECT dátum, Count(azonosító)
FROM ebéd
WHERE desszert
GROUP BY dátum
HAVING Count(azonosító)>=100;
```

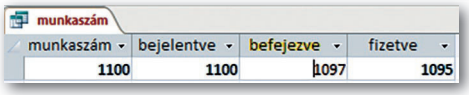
27. példa: A Null értékű mezők hatása az eredményre

A *javítás* adatbázist használva oldjuk meg az alábbi feladatokat!

- Adjuk meg, hogy hány munka adatai szerepelnek az adatbázisban! *(munkaszám)*
- Listázzuk ki, hogy az egyes években mennyi bevételt könyvelhetett el a cég, ha 5000 Ft-os óradíjjal dolgoztak! Csak azokat a munkákat vegyük figyelembe, amelyeket már ki is fizettek! *(bevételévente)*
- Határozzuk meg, mennyi volt típusonként a javítások átlagos anyagára! *(típusátlag)*



Mező:	munkaszám: id	bejelentve: bejelentés	befejezve: befejezés	fizetve: fizetés
Tábla:	munka	munka	munka	munka
Összesítés:	Count	Count	Count	Count
Rendezés:				
Megjelenítés:	☑	☑	☑	☑
Feltétel:				
vagy:				



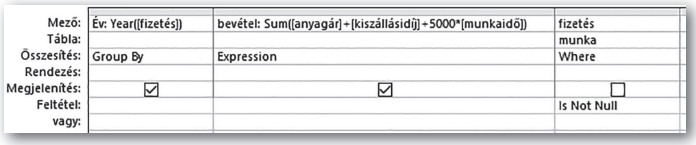
munkaszám

munkaszám	bejelentve	befejezve	fizetve
1100	1100	1097	1095

A *munkaszám* lekérdezés első ránézésre a legegyszerűbb feladatok egyike. Valóban az, de nem tanulság nélküli. Az azonosító *(id)* megszámlálása az adatbázisba bejegyzett munkák számát adja. Nézzük meg, hogy milyen eredményt kapunk, ha nem az *id*-t, hanem a *bejelentés,* a *befejezés,* valamint a *fizetés* mezőkre használjuk a COUNT függvényt! Azt látjuk, hogy a *bejelentve* mezőnél a darabszám egyezik a munkaszámmal. Ez nem meglepő, mert a munkát bejelentéskor veszik fel. A befejezett munkák száma természetesen kevesebb az összesnél, és néhány munka még a befejezettek közül sincs kifizetve. Azon mezőkbe, amelyek értéke nem ismert, nem kerül semmi, pontosabban egy úgynevezett NULL értéket vesznek fel.

> A **NULL** értékű mezőket az Access nem veszi figyelembe a számlálásnál.

A *bevételévente* lekérdezés csak annyi újdonságot hordoz, hogy a csoportosítást és az összegzést (SUM) is számított érték alapján végezzük, valamint több táblából használjuk az adatokat a számításhoz.



Mező:	Év: Year([fizetés])	bevétel: Sum([anyagár]+[kiszállásidíj]+5000*[munkaidő])	fizetés
Tábla:			munka
Összesítés:	Group By	Expression	Where
Rendezés:			
Megjelenítés:	☑	☑	☐
Feltétel:			Is Not Null
vagy:			

A *bevételévente* lekérdezés SQL-nyelven:

```
SELECT Year(fizetés) AS év,
    Sum(anyagár+kiszállásidíj+5000*munkaidő) AS bevétel
FROM munka, típus
WHERE munka.típusid=típus.id
    AND fizetés is Not NULL
GROUP BY Year(fizetés);
```

Mező:	név	átlagár: anyagár
Tábla:	típus	munka
Összesítés:	Group By	Avg
Rendezés:		
Megjelenítés:	☑	☑
Feltétel:		
vagy:		

A *típusátlag* lekérdezésben az ÁTLAG függvény (AVG) alkalmazására láttunk példát.

A **MIN()**, **MAX()**, **COUNT()**, **SUM()**, **AVG()** függvényeket **aggregáló** (csoportosító) függvényeknek nevezzük, mert ezeket általában a rekordok meghatározott csoportjaira alkalmazzuk.

Feladatok

Oldjuk meg a feladatokat a *városok* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg, milyen lélekszámú a legkisebb magyar város! *(f11a)*
b) Határozzuk meg, hogy hány fő él városban Magyarországon! *(f11b)*
c) Listázzuk ki a megyei jogú városok átlagos lélekszámát! *(f11c)*
d) Határozzuk meg, hogy hány város található Pest megyében! *(f11d)*
e) Listázzuk ki, hogy az egyes megyékben hány város van! A sorrendet a városok száma határozza meg! *(f11e)*

Segédlekérdezések

Valaha, amikor a mai adatbázisok tábláit még nem a számítógép háttértára őrizte, hanem papírra nyomtatottan használtuk őket, a legtöbb kérdésre több lépésben adtuk meg a választ. Ha meg akartuk határozni, hogy mi a székhelye annak a megyének, ahol Sarkad található, a következő utat jártuk be. Először megkerestük a települések között Sarkadot, leolvastuk a megye nevét, tehát már tudtuk, hogy Békés megye székhelyét kell megtalálnunk. Aztán a megyék táblázatában Békés megyénél megnéztük a megyeszékhely nevét. Tehát két lépés kellett ahhoz, hogy megkapjuk Békéscsabát. Valószínűleg mindenki tud példát mondani arra, amikor a kívánt információhoz – a fentihez hasonlóan – többlépéses út vezet. Sok olyan feladat van, amelyet ma, az adatbázisok korában is így oldunk meg, mert számunkra ez a megoldás természetes útja, vagy másképp nem is lehetséges.

Ha egy lekérdezés elkészítése során más lekérdezéseket is készítünk, és azokat fel is használjuk, **allekérdezésről** beszélünk. A külön elmentett lekérdezést **segédlekérdezésnek** nevezzük.

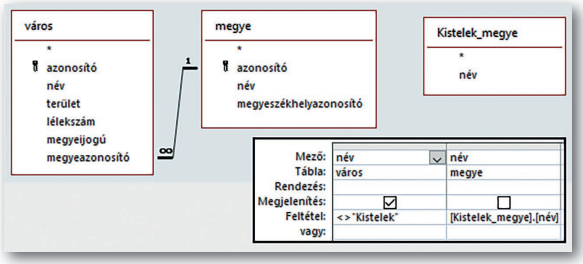
Egyes adatbázis-kezelő rendszerekben segédlekérdezések helyett úgynevezett *nézet* táblákat használhatunk. Ahogy a segédlekérdezés, úgy a *nézet* tábla is mindig az általa használt táblák aktuális állapotával dolgozik. Ezekből további lekérdezések készíthetők.

A segédlekérdezés és az allekérdezés gyakran egymás alternatívái. A segédlekérdezésekből felépülő lekérdezések általában könnyebben áttekinthetők, mint az allekérdezést használó megoldások.

28. példa: Segédlekérdezéssel megoldható problémák

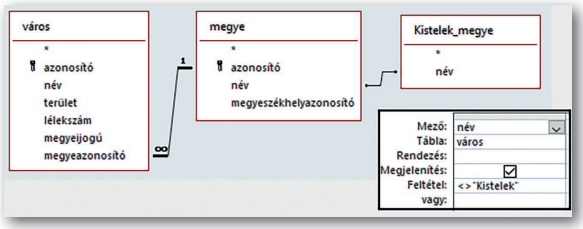
A *városok* adatbázist használva oldjuk meg az alábbi feladatokat!

- Készítsünk lekérdezést, amely megadja, mely városok találhatók ugyanabban a megyében, mint Kistelek! Kistelek nevét ne jelenítsük meg! *(Kistelek)*
- Határozzuk meg, hogy mely városok lélekszáma kisebb, mint Visegrádé! *(Visegrádnálkisebb)*
- Adjuk meg, hány város területe kisebb az átlagosnál! *(átlagnálkisebb)*
- Listázzuk ki, mely megyékben van ugyanannyi város, mint Hevesben! *(Heves)*

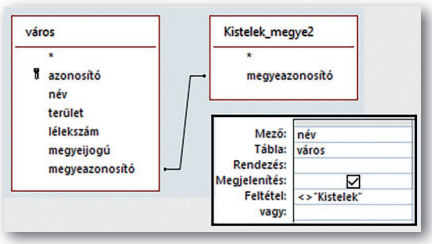


A *Kistelek* feladatra több, egymással egyenértékű megoldást mutatunk. Az első változat illeszkedik leginkább a természetes emberi gondolkodásmódhoz. Először elkészítettük a *Kistelek_megye* lekérdezést, amely két tábla felhasználásával Kistelek megyéjének nevét adja meg. Az ábrán látható, hogy a válaszadáshoz készített lekérdezésbe nem csupán a két táblát, hanem a *Kistelek_megye* lekérdezést is felvettük mint segédlekérdezést. Két feltételt adtunk meg. Az egyikkel Kistelek megyéjének nevére szűrünk. Ha a feltétel sorában egy lekérdezés vagy tábla egy mező-

jét adjuk meg, akkor a tábla/lekérdezés és a mező nevét is szögletes zárójelbe kell írnunk, és a kettő közé pontot tennünk. A másik szűrési feltétellel pedig kizárjuk a megjelenítendő városok közül Kisteleket.



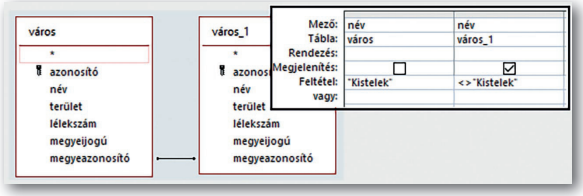
Az előző megoldás jól követhető és logikus, azonban érdemes egy kicsit módosítani, hogy egyszerűbbé váljon. Vegyük észre, hogy a megye nevére vonatkozó feltétel pontosan a *Kistelek_megye* lekérdezés és a *megye* tábla közötti kapcsolat. Ezt grafikusan is megjeleníthetjük a mellékelt ábrának megfelelően.

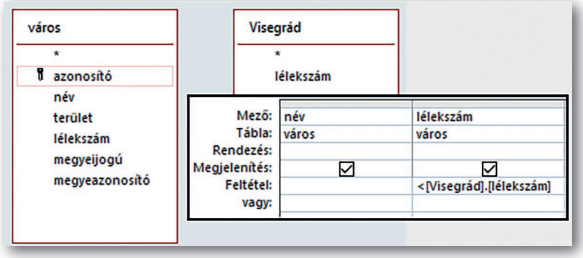


Ha alaposan megnézzük az előző megoldásokat, észrevehetjük, hogy a megye neve nem is lényeges a feladat szempontjából. A lényeg, hogy ugyanabban a megyében legyen a város. A megyét a *város* táblában található *megyeazonosító* meghatározza, tehát a megoldás egyszerűsíthető a következő módon: adjuk meg azokat a városokat, amelyek *megyeazonosítója* ugyanaz, mint Kisteleké! Ezt a megoldást természetesen olyan lekérdezéssel kell előkészítenünk, amely a megye neve helyett a megye azonosítóját adja. Ez a *Kistelek_megye2* lekérdezés, amely csak egyetlen táblát használ. Maga a lekérdezés az ábrán látható formára egyszerűsödik.

A *Kistelek* lekérdezés harmadik megoldásának SQL-nyelvű változata, amely segédlekérdezés helyett allekérdezést használ:

```
SELECT név
FROM város, (SELECT megyeazonosito FROM város WHERE név=”Kis-
  telek”) AS Kistelek_megye
WHERE város.megyeazonosito=Kistelek_megye.megyeazonosito
  AND név<>”Kistelek”;
```



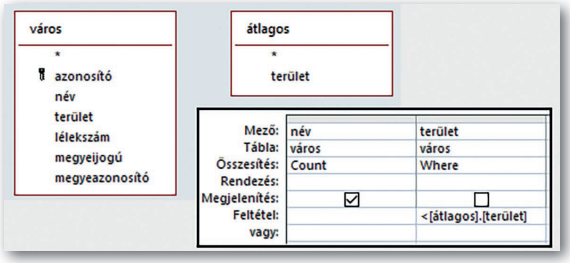


Az előző három megoldás mindegyike egy lekérdezéssel készítette elő a kérdés megválaszolását. Létezik olyan megoldás is, amely nem igényel plusz lekérdezést, de a *város* tábla többszöri felhasználását igen. Ez a legközvetlenebb megoldás, hasonlóval már találkoztunk korábban is, de ezt a formát kevesen szokták használni logikai összetettsége miatt.

A *Visegrádnálkisebb* lekérdezéshez szükségünk van egy Visegrád lélekszámát meghatározó lekérdezésre. Ezt használjuk fel a következő lépésben.

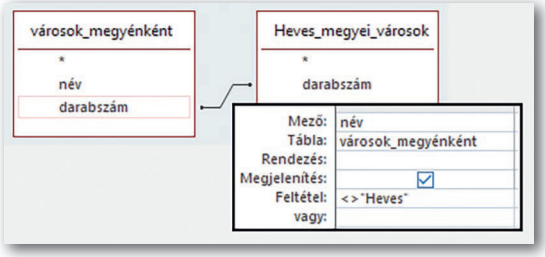
Itt nincs mód összekapcsolni egy másik mezővel, hiszen nem egyenlő, hanem kisebb reláció van a keresett városok és a meghatározott szám között. Az ábrán látható feltételt begépelhetjük, ekkor vigyáznunk kell, hogy az összes zárójel a helyén legyen, vagy a *helyi menü* > *Szerkesztés* pontját választva is megalkothatjuk.

Az *átlagnálkisebb* lekérdezés elkészítéséhez szükségünk van a települések átlagos területére. Nyilván ehhez az értékhez kell viszonyítani az egyes települések területét. A tábla és az átlag értékét adó lekérdezés között nincs kapcsolat, a lekérdezőrácson tudjuk beállítani a szükséges feltételt. Mivel nem a kimenet eredményére szűrünk, ezért az összesítés sorában a WHERE-t kell beállítanunk.



Az *átlagnálkisebb* lekérdezés megoldásának SQL-nyelvű változata, amely segédlekérdezés helyett allekérdezést használ:

```
SELECT Count(név)
FROM város
WHERE terület<(SELECT AVG(terület) FROM város);
```



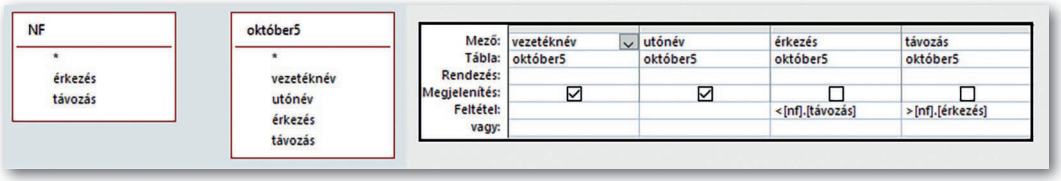
A *Heves* lekérdezés előkészítéséhez két lekérdezést is praktikus létrehoznunk. Az egyikben meghatározzuk a Heves megyei városok számát, a másikban pedig azt, hogy az egyes megyékben hány város van. Mivel az egyező darabszám a kérdés, ezért ezt a két darabszám mező közötti kapcsolat megadásával is beállíthatjuk.

29. példa: Táblák (és lekérdezések) között többszörös kapcsolat

Az *étkezés* adatbázist használva oldjuk meg az alábbi feladatokat!

- Készítsünk lekérdezést, amely megadja, kikkel találkozhatott ebéd közben Nagy Ferenc 2020. 10. 05-én! *(találkozás)*
- Határozzuk meg, hogy mely napokon hányan ették ugyanazt a menüt, és választottak ugyanúgy desszertet, mint Németh Sára! *(egyezően)*
- Az előzőhöz hasonlóan határozzuk meg, hogy mely napokon hányan ették ugyanazt a menüt, és választottak ugyanúgy desszertet, mint Németh Sára! Németh Sárát ne számítsuk közéjük! *(nélkül)*

A *találkozás* lekérdezés elkészítéséhez úgy jutunk közelebb, ha tudjuk Nagy Ferenc 2020. 10. 05-i érkezési és távozási időpontját. Ezt az *NF* lekérdezés adja meg. Az *október5* lekérdezés kilistázza az összes, 2020. 10. 05-én ott ebédelő érkezési és távozási adatait. A két

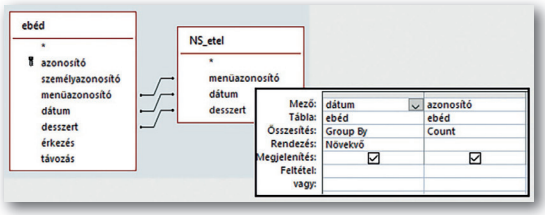


segédlekérdezés segített megszabadulni a probléma szempontjából érdektelen adatoktól, így már koncentrálhatunk csak a lényegre.

Gondoljuk végig az ábra segítségével, hogy miképpen helyezkedhetnek el az egyes vendégek jelenléti intervallumai Nagy Ferenc érkezési és távozási időpontjához képest!

A piros és kék vonalak ábrázolják két személy jelenlétét egy eseményen. A vonal kezdete az érkezés időpontját, a vége a távozását jelzi. Ha a két vonal legalább részben fedésben van, akkor találkoztak. Találkozásra csak akkor kerül sor, ha a vonalak helyzete egyezik a fenti négy egyikével. Miképpen tudjuk leírni a találkozást az érkezési és távozási időpontokkal? Mikor találkozott a piros a kékkel? Ehhez szükséges, hogy a piros hamarabb érkezzen, mint ahogy a kék távozik, de kell az is, hogy a piros később távozzon, mint ahogy a kék érkezik. Ennek a két feltételnek együtt kell teljesülnie. Ez a találkozással kapcsolatos feltétel legegyszerűbb megfogalmazása.

A fenti meggondolás alapján megfogalmazhatjuk a két segédlekérdezésen alapuló, megoldást adó feltételt. Ez a megoldás több más, intervallumok fedésén alapuló probléma leküzdéséhez is utat mutathat.

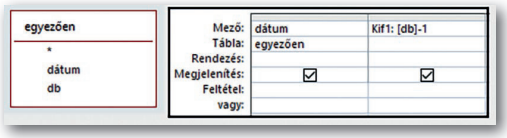


Az *egyezően* lekérdezéshez csak egy segédlekérdezést készítettünk. Az *NS_étel* lekérdezés megadja, hogy melyik napon melyik menüt ette Németh Sára, és azt is, hogy fogyasztott-e hozzá desszertet. A főlekérdezésben a segédlekérdezés kimeneti mezőit használjuk fel. Mivel ugyanakkor és ugyanazt kell enniük a keresett személyeknek, mint Németh Sárának, ezért a megfelelő mezők egyezésének teljesülnie kell. Ezt beállíthatjuk a lekérdezőrácson, de megadhatjuk az *ebéd* tábla és az *NS_étel* lekérdezés közötti kapcsolatok segítségével is. Itt az utóbbit választottuk. A rács pusztán azt írja le, hogy az egyes napok hányszor fordulnak elő. Fontos megjegyezni, hogy ez a lekérdezés olyan eredményt ad, amelyben Németh Sára is szerepel.

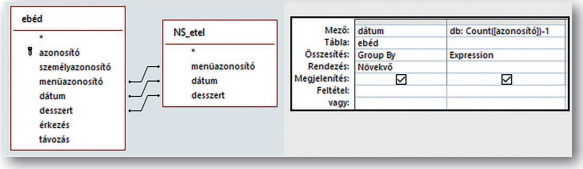
Az *egyezően* lekérdezés megoldásának SQL-nyelvű változata, amely segédlekérdezés helyett allekérdezést használ:

```
SELECT ebéd.dátum, Count(azonosító)
FROM ebéd,
    (SELECT menüazonosító, dátum, desszert
          FROM vendég, ebéd
          WHERE vendég.azonosító=ebéd.személyazonosító
                AND vezetéknév= ”Németh”
                AND utónév= ”Sára”
    ) AS NS_étel,
WHERE ebéd.dátum=NS_étel.dátum
    AND ebéd.menüazonosító=NS_étel.menüazonosító
    AND ebéd.desszert=NS_étel.desszert
GROUP BY ebéd.dátum;
```

A *nélkül* lekérdezést látszólag nagyon egyszerű elkészíteni, hiszen egyszerűen le kell vonni egyet az előző eredmény minden sorából. Ezt a matematikai műveletet nem alkalmazhatjuk közvetlenül a lekérdezőrácson.



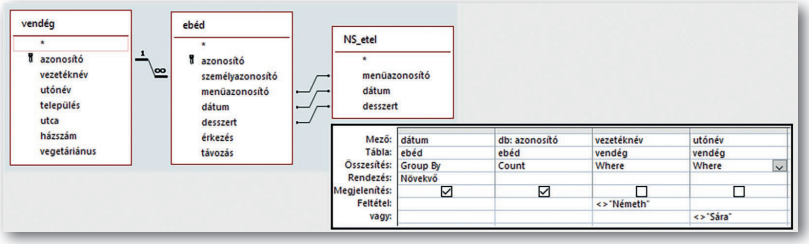
A lekérdezés elkészítésének legegyszerűbb módja, ha az előbb elkészített *egyezően* lekérdezést mint segédlekérdezést használjuk. Ekkor csak annyi a teendőnk, hogy minden ott kapott számértékből kivonunk egyet.



A másik lehetőség az *egyezően* lekérdezés módosításával adódik, de ehhez a mező sorában szerepelő kifejezést, amely a COUNT függvénynél eggyel kisebbet ad eredményül, nekünk kell begépelnünk. Ha az *Összesítés* gomb be van kapcsolva, és a mező sorában nem mezőnév szerepel, az *Összesítés* sorában általában az *Expression* szót kell kiválasztanunk a listából.

A harmadik lehetőség is egészen kézenfekvőnek tűnik: a megszámláltak közül ki kell zárnunk Németh Sárát. Ehhez az *egyezően* lekérdezést kell kibővítenünk a *vendég* tábla felvételével. A lekérdezőrácson kell úgy módosítanunk, hogy Németh Sárát ne számlálja. A feltétel megfogalmazásához vissza kell nyúlni a matematikai logikában tanultakhoz, mely szerint a Nem(A és B) kifejezés egyenértékű a Nem(A) vagy Nem(B) kifejezéssel.

Érdekes volt megtapasztalni, hogy egyetlen apró kis változtatás a feladatban jelentős módosítást igényel a megoldásban.



Feladatok

Oldjuk meg a feladatokat a *javítás* adatbázist használva, lekérdezőrács segítségével!

a) Adjuk meg, kik laknak ugyanazon a településen, mint Balogh Attila! *(f12a)*
b) Listázzuk ki, hogy kik fordultak ugyanolyan típusú problémával a céghez, mint Kovács Emma 2019-ben! *(f12b)*
c) Határozzuk meg, hány számlán szerepelt az átlag kétszeresénél nagyobb anyagköltség 2019-ben! *(f12c)*
d) Adjuk meg, hogy melyek azok a hibatípusok, amelyek a villannyal kapcsolatos problémáknál többször fordulnak elő! *(f12d)*
e) Listázzuk ki, hogy mely hibatípusok átlagos időszükséglete nagyobb, mint a klímával kapcsolatos hibák javítási idejének átlaga! *(f12e)*

Mi van, ha nincs?

Az eddig tárgyalt feladatok mindegyike arra vonatkozott, amit az adatbázis tábláiban közvetlenül megtaláltunk. Ritkábban bár, de előfordulnak olyan kérdések is, amelyek pontosan arra vonatkoznak, ami nem szerepel. Az *étkezés* adatbázisban rögzítették azt, hogy ki ebédelt 2020. 10. 05-én, de nem szerepel, hogy ki nem ebédelt akkor. Természetesen tudunk válaszolni erre a kérdésre, de csak azzal a megszorítással, hogy az adatbázisban szereplő személyek közül ki nem ebédelt. Gondoljunk vissza arra, hogy mit tanultunk a halmazokról matematikából! Akármiről is beszéltünk, mindig volt egy alaphalmaz. Bármely halmaz került szóba, az ennek az alaphalmaznak egy részhalmaza volt. Értelmeztük a halmaz kiegészítő halmazát, azaz komplementerét. Itt az összes vendég számít alaphalmaznak. Az ebédelők a részhalmaz, amelyet tárolunk, és ennek komplementerére vagyunk kíváncsiak. Tehát adatbázis-kezelésből ezen komplementerhalmaz előállítását kell megismernünk.

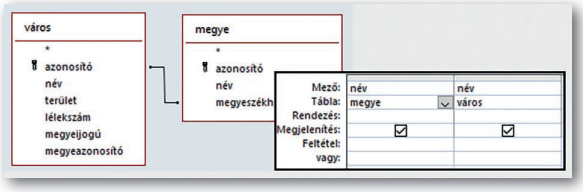
A legnagyobb nehézséget nem az jelenti majd, hogy miképpen készítjük a komplementerhalmazt, hanem annak felismerése, hogy mikor van rá szükségünk.

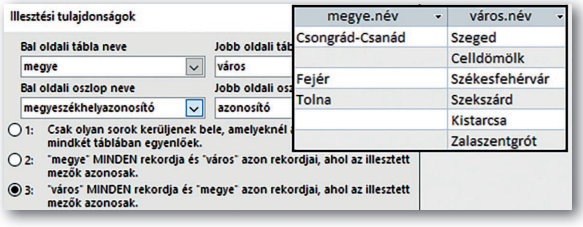
30. példa: Hiányzó adat keresése

A *városok* adatbázist használva oldjuk meg az alábbi feladatokat!

- Adjuk meg azokat a városokat, amelyek nem megyeszékhelyek! *(nemszékhely)*
- Melyek azok a megyék, amelyekben nincs 50 ezer főnél népesebb város? *(nincsnagy)*

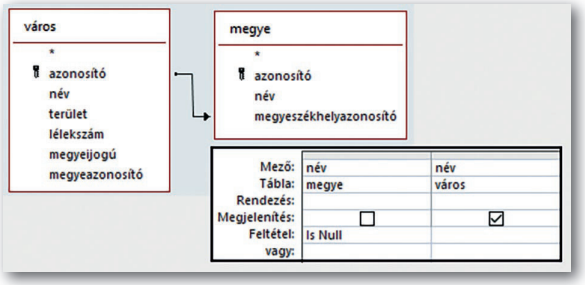
A *nemszékhely* lekérdezés kapcsán azt nézzük meg, ami van, és nem azt, ami nincs. A *megye* táblában a *megyeszékhelyazonosító* adja meg azt a *várost,* amely a megyeszékhely. Ha a *megye* és a *város* táblák közötti kapcsolatot közöttük hozzuk létre, akkor nem minden városhoz kapcsolódik megye, és az eredményhalmazba 19 rekord tartozik majd.





Ezt a *megyeszékhely* lekérdezést fogjuk majd átalakítani a cél érdekében. Kattintsunk duplán a két táblát összekötő, kapcsolatot jelző vonalra! A megjelenő párbeszédablakban a két tábla közötti kapcsolatot három különböző módon állíthatjuk be. Ha nem módosítjuk az alapértelmezést (1-es), csak azok a rekordok jelennek meg, amelyekben a két összekapcsolt mezőnek egyezik az értéke. Ha a 3-ast választjuk, akkor jelen esetben úgy hozza létre a kapcsolatot, ha a *város* tábla minden elemét felsorolja a *város.név* oszlopban, mellé pedig akkor írja be a megye nevét, ha az a megyeszékhely, egyébként a mező üresen marad. A lekérdezés eredménye látszik a képen is.

A *nemszékhely* lekérdezés végleges formáját a következő oldali ábra mutatja. A kapcsolati rajzból leolvasható, hogy a *város* tábla minden eleméhez próbál értéket társítani a *megye* táblából. Fentebb láttuk, hogy a megyeszékhelyek esetén olvasható a megye



neve, egyébként a mező üres. Az üres mezőkre az **Is Null** kifejezéssel tudunk szűrni, a nem üresekre pedig az **is Not Null** kifejezéssel. Most az előbbit használjuk a feladat megoldása érdekében.

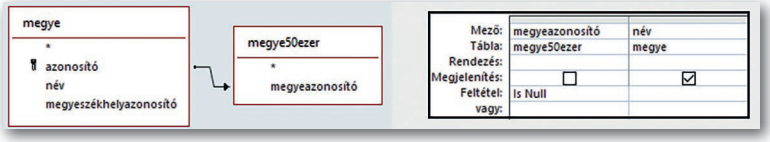
A *nemszékhely* lekérdezés SQL-nyelvi alakja:

```
SELECT város.név
FROM város LEFT JOIN megye ON megye.megyeszékhelyazonosító =
  város.azonosító
WHERE megye.név Is Null;
```

A *nemszékhely* lekérdezést elkészíthetjük allekérdezéssel is:

```
SELECT név
FROM város
WHERE azonosító NOT IN (SELECT megyeszékhelyazonosító FROM megye);
```

Az allekérdezéses megoldás fejezi ki a legszemléletesebb módon a bevezetőben ismertetett gondolatmenetet. Úgy fordíthatjuk hétköznapi nyelvre, hogy kilistázzuk azon városok nevét, amelyeknek az *azonosítója* nem szerepel a *megye* táblában tárolt *megyeszékhely-azonosítók* között.



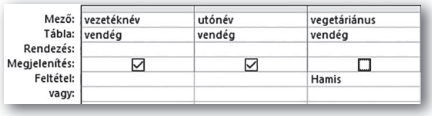
A *nincsnagy* lekérdezést közvetlenül, két táblát használva nem tudjuk elkészíteni. Abból csak azt tudnánk meghatározni, hogy melyikben van legfeljebb 50 ezer fős város. Az viszont nem jelenti azt, hogy nincs nagyobb város. Viszont ha elkészítjük az 50 ezer főnél népesebb városok megyéinek listáját, azt segédlekérdezésként fel tudjuk használni a *nemszékhely* lekérdezésnél megismert formában.

31. példa: Összetett lekérdezések

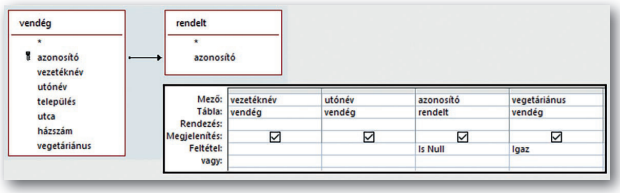
Az *étkezés* adatbázist használva oldjuk meg az alábbi feladatokat! Vajon melyik igényli a most szerzett ismereteket, és melyiket tudtuk volna már korábban megoldani?

- Készítsünk lekérdezést, amely megadja, hogy kik nem vegetáriánusok! *(nemvegetáriánus)*
- Határozzuk meg, hogy a vegetáriánusok közül ki nem rendelt vegetáriánustól különböző menüt! *(nemrendelt)*
- Adjuk meg lekérdezés segítségével, hogy mely napokon fogyott legfeljebb 80 desszert! *(max80)*

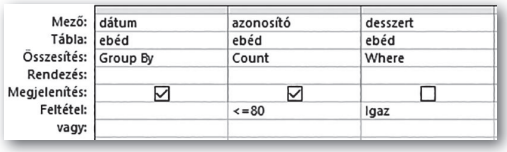
Alapos végiggondolás után a legtöbben azt mondják, hogy a *nemvegetáriánus* és *max80* lekérdezések azok, amelyeket korábban is meg tudtunk volna oldani. Lássuk, valóban így van-e!



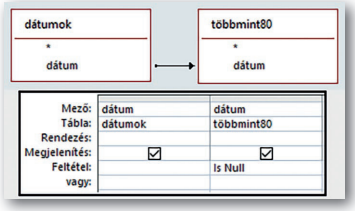
A *nemvegetáriánus* lekérdezéshez valóban nem kell pluszismeret, mivel a *vendég* tábla tartalmazza ezt a tulajdonságot. Így, ha a *vegetáriánus* értéke hamis, az illetőt meg kell jeleníteni. Mikor lett volna szükségünk a frissen tanultakra? Ha a vegetáriánusokat – alapvetően ezt a tulajdonságot – egy külön tábla tartalmazza, akkor a probléma hasonló lenne a *nemszékhely* feladathoz.



A *nemrendelt* lekérdezésnél – az előzőleg alkalmazott módszer szerint – elkészítjük azt a lekérdezést, amely megadja, kik azok, akik vegetáriánusként rendeltek nem vegetáriánus menüt. Ezt a lekérdezést mint segédlekérdezést felhasználjuk a megoldásban, mégpedig a *vendég* táblához illesztve. A *vendég* táblából mindenki szerepel, a *rendelt* lekérdezésből pedig azok, akiknél a megadott kapcsolatnak megfelelően van vele egyező érték. Ahol nincsenek megjelenített értékek, azokat kell megadnia a lekérdezésnek. Ezen rekordokra az **Is Null** feltétellel szűrünk a rácson – ahogyan az ábra is mutatja.



A *max80* lekérdezés elkészítése nem látszik túlzottan bonyolultnak. Egyetlen tábla segítségével válaszolhatunk: naponkénti csoportosításban megszámláljuk a desszertet fogyasztókat. Ha számuk nem haladja meg a 80 főt, akkor megjelenítjük. Mi ezzel a probléma? Látszólag semmi. Próbáljunk ki valamit! Jegyezzünk be egy újabb rekordot az *ebéd* táblába 2020. 11. 02-ára! Az 1-es azonosítójú személy C menüt rendelt, és nem kért desszertet. Az érkezés és távozás idejét állítsuk be tetszőlegesen! Futtassuk le az előző lekérdezést! Az eredmény 2020. 11. 02-át nem tartalmazza, pedig aznap csak ezt az egy ebédet fogyasztották, és így biztosan kevesebben fogyasztottak desszertet 80 főnél. Miért nem szerepel ez a dátum? Mert megszámlálni csak azt lehet, ami volt. Aznap pedig nem volt desszertet fogyasztó.



Hogyan oldható meg ez a probléma? Két segédlekérdezést kell készítenünk. Az egyiknek meg kell adnia az összes dátumot *(dátumok)*, a másiknak *(többmint80)* pedig felsorolnia, hogy mely napokon volt több mint 80 fő desszertet fogyasztó. Végül a *dátumok* lekérdezésből – a megszokott módon – meg kell jelenítenünk azokat a napokat, amelyek nem szerepelnek a *többmint80* lekérdezés kimenetében.

Feladatok

Oldjuk meg a következő feladatokat lekérdezőrács segítségével! Van-e olyan, amelyet a most tanult ismeretek nélkül is meg lehet oldani?

a) A *javítás* adatbázisban tároltak alapján adjuk meg, kik laknak ugyanazon a településen, mint Balogh Attila! *(f13a)*
b) A *javítás* adatbázis alapján adjuk meg, milyen típusú javításokat végeztetett Katona Lara, amiket Mezei Fanni nem! *(f13b)*
c) Az *étkezés* adatbázis használatával adjuk meg, kik azok, akik soha nem ettek húsimádó menüt! *(f13c)*
d) Az *étkezés* adatbázis alapján állapítsuk meg, milyen menüt nem választott soha Barna Panna! *(f13d)*
e) Az *étkezés* adatbázisban tároltak alapján adjuk meg, kikkel nem találkozott Nagy Dávid 2020. 10. 20-án ebéd közben annak ellenére, hogy aznap ők is ott ebédeltek! *(f13e)*

Jelentések

A lekérdezések fő célja, hogy a tárolt adatok alapján információhoz jussunk. Nem érdekes, hogy milyen formában jelenik meg, hanem csak a tartalma. Azonban van, amikor fontos, hogy az előállított adatokat könnyen áttekinthetően, a lényeget kiemelve jelenítsük meg. Az eszközt, amellyel ezt a célt elérhetjük, jelentésnek nevezzük.

A **jelentés** az adatbázisban tárolt adatoknak és az azokból kinyerhető információknak az áttekinthető megjelenítésére szolgál – gyakran nyomtatott formában.

Jelentést készíthetünk táblából vagy lekérdezésből. Jelentés készülhet több táblából is, ha az Adatbáziseszközök Kapcsolatok lapján megadtuk a köztük lévő kapcsolatokat.

32. példa: Egyszerű jelentés egy tábla alapján

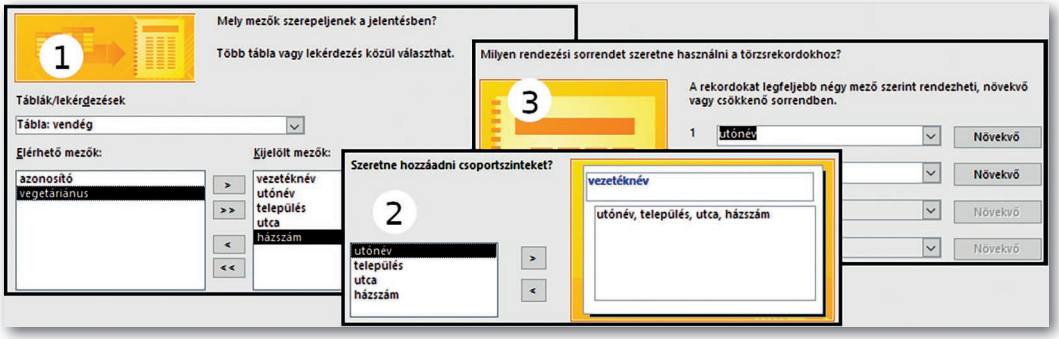
Készítsünk jelentést az *étkezés* adatbázis *vendég* táblájából az ábrán látható minta alapján! A személyeket csoportosítsuk vezetéknév szerint, azon belül pedig rendezzük az utónevük alapján! Jelenítsük meg a címüket is! Az elkészítés során ügyeljünk arra, hogy minden adat teljes egészében látható legyen! A fejrész tartalma ékezethelyesen jelenjen meg! Törekedjünk a mintán látható formai jellemzők kialakítására is! A kész jelentést őrizzük meg fájlba nyomtatva is! *(vendéglista)*

A vendégek listája betűrendben

vezetéknév	utónév	település	utca	házszám
Bakos				
	Gergely	Felsőváros	Rakéta utca	27
	Olivér	Nagyfalu	Rezeda dűlő	39
Balog				
	Lilien	Kisfalu	József utca	17
	Noel	Felsőváros	Verseny utca	57
Balogh				
	Adél	Alsóváros	Pellérdi út	21
	Eszter	Felsőváros	Görgey Artúr utca	4
	Hunor	Nagyfalu	Stiglicfogdosó	32
	Levente	Nagyfalu	Kút utca	3

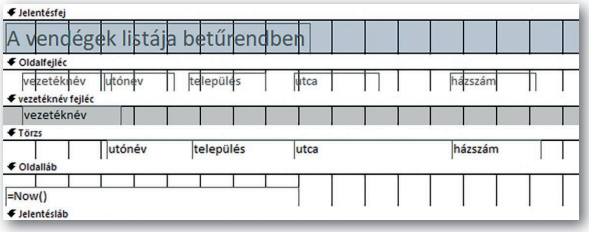
A folyamatot a *Létrehozás* > *Jelentések* > *Jelentés* varázsló elemre kattintva indíthatjuk el.

A *vendéglista* jelentés egyetlen adattáblából készül.



A varázslóban megtett lépések közül a fontosabbak láthatók az ábrán. (1) Kiválasztjuk a táblát, és megjelöljük azokat a mezőket, amelyeket felhasználunk. (2) Megadjuk, hogy mely mező vagy mezők szerint csoportosítunk. (3) Beállítjuk a sorba rendezés kulcsait és irányát. Ezeket nagyon könnyen megtehetjük a minta alapján. A folyamat további lépéseinél az elrendezésről, a grafikai és tipográfiai beállításokról döntsünk belátásunk szerint!

A program által előállított végeredmény gyakran elmarad elvárásainktól. Elképzelhető, hogy az egyes adatok távolsága a soron belül jóval nagyobb lesz, más adatok viszont nem is látszanak teljes egészében. A formai beállításokat az *Elrendezési* vagy a *Tervező*



nézetben módosíthatjuk. Ha még nem vagyunk rutinosak, az *Elrendezési nézetet* válasszuk, abban jobban látjuk változtatásaink hatását. A *Tervező nézet* – amellyel akár létre is hozhatjuk a jelentést – segít áttekinteni annak szerkezetét, amely az alábbi ábrán látható.

A jelentés részei:
Jelentésfej: A jelentésben egyszer, a legelején jelenik meg, tartalma állandó.
Oldalfejléc: Minden oldal tetején olvasható, általában a jelentés oszlopainak azonosítását szolgálja.
Csoportfejléc: Minden csoport kezdetén megtalálható, tartalma a csoportot alkotó tulajdonság értéke.
Törzs: A jelentésbe felvett tulajdonságokat tartalmazza – a csoportfejléc adatait kivéve.
Csoportlábléc: Általában a csoporthoz tartozó összegzést, a rekordszámot, a számokat tartalmazó mezők összegét, átlagát, minimumát, maximumát tartalmazza. (Ez az ábrán nem látható, mert ebben a feladatban üres.)
Oldalláb: Minden oldal alján olvasható szöveg, jellemzően dátum, időpont, oldalszám szerepel benne.
Jelentésláb: A jelentésben egyszer, a végén olvasható, tartalma gyakran összegzés.

Váltsunk át a *Tervező nézetre,* és vessük össze az itt látható képet a mintaként adottal!

- Állítsuk be az egyes mezők méretét úgy, hogy a tárolt adatok elférjenek bennük, de ne legyenek túl szélesek! Figyeljünk arra, hogy a fejlécben az oszlopnevek méretét is módosítsuk!
- Változtassuk meg az egyes mezők helyét úgy, hogy ne legyen köztük túl nagy távolság! A pozíció beállítását kövesse az oszlopnevek helye is!
- Írjuk át az oszlopnevek tartalmát a mintának megfelelően! Ha nem áll rendelkezésre minta, legyen tartalmilag kifejező és ékezethelyes!
- Módosítsuk a jelentésfejet a minta szerint! Ha nincs előírva a tartalma, akkor válasszunk rövid, kifejező címet!

33. példa: Jelentés több tábla felhasználásával

Több táblát felhasználva is készülhet jelentés, ha a táblák kapcsolatát az adatbázisban megadtuk.

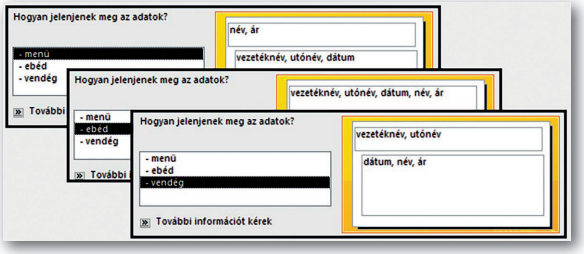
Készítsünk jelentést az *étkezés* adatbázis tábláiból a következő oldali ábrán látható minta alapján!

- Jelenítsük meg, hogy melyik napon ki milyen menüt evett!
- Az adatokat csoportosítsuk dátum szerint, a személyeket jelenítsük meg vezetéknév, azon belül utónév szerinti sorrendben!
- Tüntessük fel a menü nevét és árát is! A számértéket állítsuk be nulla tizedesjegy pontosságra, mögötte jelenítsük meg a pénznemet is!
- Minden napra adjuk meg az aznapi ebédelők számát és az ebédből származó bevételt!

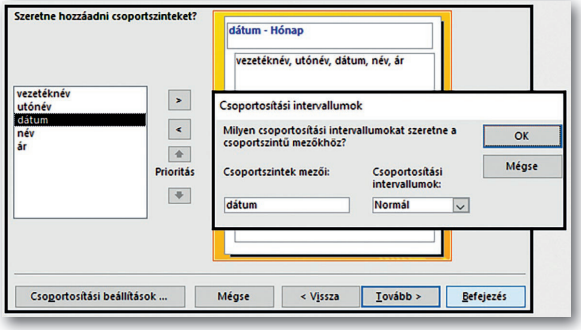
Nézzük, hogy milyen újdonságokat tartalmaz ez a feladat!

Az (1) lépésben egymás után több táblát kell felvennünk, és táblánként a megfelelő mezőket választanunk.

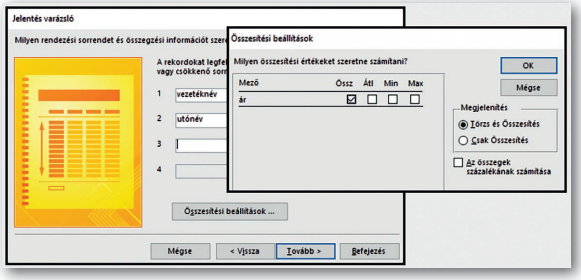
A (2), csoportképzést megadó lépés előtt a felhasznált táblákhoz és az adatbázis szerkezetéhez illeszkedő beállítások – nézetek – közül választhatunk. Az ábrán is jól látható a táblák kapcsolata: ha a menü oldaláról szemléljük, akkor egy menühöz több ebéd tartozik, ezért a menüt választva a program automatikusan megjelöli a csoportosítás alapjának. Ugyanígy történik a vendég választásakor, mivel egy vendéghez szintén több ebéd tartozhat. Ha az ebédet jelöljük meg, akkor nem kínál fel automatikusan *csoportszintet,* mert egy ebédhez pontosan egy vendég és pontosan egy menü tartozik. Mivel nekünk *dátum* szerint kell csoportosítanunk, ezért e lehetőségek közül válasszuk az ebédet, majd a következő lépésben a *dátumot* jelöljük meg a csoportosítás alapjának! Ha dátum típusú a *csoportszint,* az Access alapértelmezésben havi csoportosítást állít be. Ezt a *Csoportosítási beállítások* gombra kattintva céljainknak megfelelően tudjuk módosítani. A napi csoportosításhoz válasszuk a *Normál* beállítási lehetőséget! Szöveg típusú adatnál a teljes tartalom helyett kezdőbetű(k) szerint is csoportosíthatunk.



▶ Előzetes csoportbeállítás

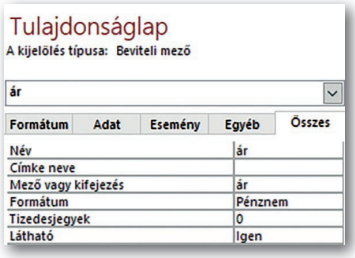


▶ Csoportosítási beállítások



▶ Összesítési beállítások

A (3), rendezési lépésben mód van arra is, hogy a *vezetéknév* és *utónév* szerinti rendezés beállításával együtt az adott csoportra vonatkozóan összesítést készítsünk. Ennél a jelentésnél az *ár* mező összegzését végezzük el.



Az elkészült jelentést az előző feladatnál látottak alapján módosíthatjuk úgy, hogy a mezők elhelyezése és a fejek beállítása megfeleljen a mintának. *Tervező nézetben* törölhetjük az összesítő sor felesleges elemeit is. Ezek után a jelentés egyedüli hiányossága, hogy az *ár* mező formailag eltér a mintától.

Tervező nézetben kattintsunk a *Törzs* részben az *ár* mezőre, majd jelenítsük meg a *Tulajdonságlapját,* azután állítsuk be a *Formátumot* és a *Tizedesjegyek* számát! Ugyanezt tegyük meg a *dátum láblécben* az összegző függvénnyel!

Ezt a feladatot úgy is megoldhattuk volna, hogy készítünk egy lekérdezést, amely az összes szükséges mező értékét tartalmazza, majd ezt felhasználva hozzuk létre a jelentést.

Ha a teljes adathalmaz megadott mezőinek kell a jelentésben szerepelniük, általában nincs szükségünk arra, hogy lekérdezést készítsünk. Ha nem minden adatot használunk fel, vagy meg kell jelenítenünk számított értéket is, akkor lekérdezéssel készítsük elő a jelentést!

- Készítsük el az első jelentést úgy, hogy csak az alsóvárosiak szerepeljenek!
- Készítsük el a második jelentést úgy, hogy csak október első hét napját vegyük figyelembe!

Feladatok

Készítsük el a következő jelentéseket a *javítás* adatbázist használva! Ha szükséges, lekérdezéssel készítsük elő a megoldást! A jelentéseket formailag magunk tervezzük meg!

a) Jelenítsük meg a felsővárosi lakosok nevét és e-mail-címét nevük kezdőbetűje szerinti csoportosításban! *(f14a)*

b) A munkavégzés települése és azon belüli helye szerinti csoportosításban jelenítsük meg a munka típusát és befejezésének dátumát! *(f14b)*

c) A munka típusa szerinti csoportosításban jelenítsük meg a munkavégzés helyét, dátumát, a munkaórák számát és az anyagköltséget! Csoportonként jelenjen meg a számértékek átlaga! Állítsuk be a pénznem formátumot az anyagköltségek megjelenítéséhez! *(f14c)*

Adatbevitel, űrlapok

Az adatbázis-kezelés gyakorlati részének elején létrehoztunk adatbázist, és vittünk be adatokat. Tisztáztuk, hogy nemcsak az rögzít adatokat, akinek ez a munkája, hanem az is, aki egyszerű használója egy ilyen rendszernek. Az adatbevitel során közvetlenül az adattáblába írtunk, de az egyrészt nem biztonságos, másrészt nem különösebben felhasználóbarát, elég csupán az idegen kulcs értékének bejegyzésére gondolnunk. Még kevésbé tarthatjuk megfelelőnek, hogy az egyszerű felhasználó, aki egy webáruházból rendel, ugyanilyen adattáblákat töltsön ki.

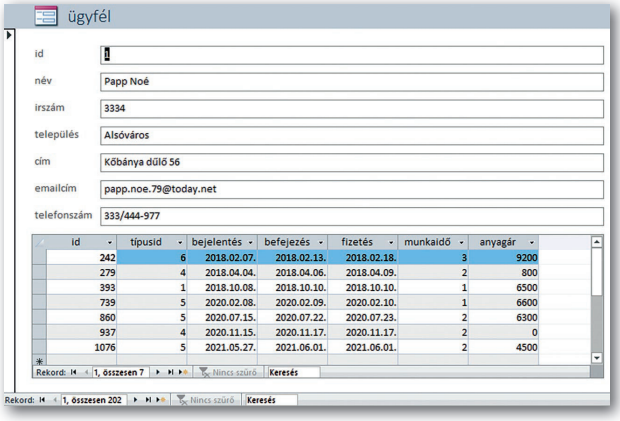
Azt a jól átlátható, könnyen kezelhető felületet, amelynek segítségével adatokat viheténk be, vagy adatokat kérdezhetünk le, **űrlapnak** nevezzük.

Az űrlapok készítését nem tekintjük mindenki számára szükséges tudásnak, használatukat viszont igen, ezért érdemes megismerni, mi van egy működő űrlap hátterében. Példáink a *javítás* adatbázishoz készültek.

34. példa: Űrlap generálása egy táblához

Készítsünk űrlapot az *ügyfél* táblához!

Az űrlapkészítés legegyszerűbb módja, hogy kiválasztjuk a megfelelő táblát, és rákattintunk a *Létrehozás* > *Űrlapok* > *Űrlap* elemre. Ekkor az Access automatikusan generál egy űrlapot, amelynek segítségével végiglépdelhetünk a tábla rekordjain, módosíthatjuk az aktuális rekordot, vagy éppen újat vehetünk fel. Mivel a *munka* táblában idegen kulcsként szerepel az ügyfél azonosítója, ezért segédűrlapként mindig megjelenik a *munka* tábla aktuális ügyfélhez tartozó része is.



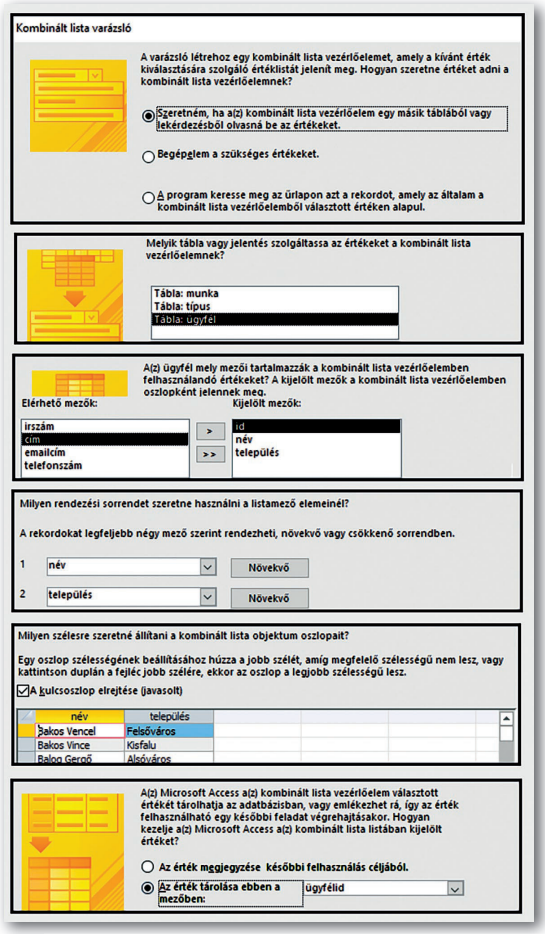
A kép alsó széléről leolvasható, hogy az *ügyfél* tábla első adatsorát látjuk a 202 rekordból. A táblázatos segédűrlap mutatja, hogy Papp Noé milyen munkákat végeztetett el. A munkának sajnos csak a *típusazonosítója* és nem a típusa szerepel, ezért nehezen használható. Az automatikusan generált űrlapoknál ilyen gyakran előfordul.

Nagyon fontos, hogy az adatbázisba ne kerüljön be hibás érték. Ezt természetesen nem lehet teljes mértékben megakadályozni, de elvárható, hogy az űrlap segítse a pontos adatrögzítést. Ennek első lépcsőfoka, hogy figyelmeztessen a nyilvánvalóan érvénytelen értékekre. Az ügyfél adatainál például az irányítószámnak 4 karakterből kell állnia. A képen látható *beviteli maszk*, az *érvényességi szabály* és az *érvényesítési*

szöveg erre vonatkozó beállítás. Ezt nemcsak az űrlap szintjén, hanem az adattábla mezőinél is be lehet állítani. Ilyen egyszerűen megfogalmazható feltétel a településnél nincs, ott egy, a létező települések nevét tartalmazó legördülő listával biztosíthatnánk, hogy csak létező település szerepeljen, és a név helyes legyen.

35. példa: Űrlap generálása segédűrlappal

Készítsünk űrlapot a *munka* táblához!



Az előző módszerrel előállított űrlapon már nem szerepel segédűrlap, és nem csupán a típus, hanem az ügyfél is azonosítóval található meg rajta. Ez az űrlaptartalom már a használhatóságot is megkérdőjelezi, mivel az ügyfeleket azonosító alapján ismerni lehetetlen. Felhasználói szemszögből az lenne az ideális, ha egy legördülő listából választhatnánk ki ezeket az értékeket. *Tervező nézetben* az *Űrlaptervezés* > *Vezérlők* > *Beviteli lista* elemére kattintva az űrlapon kattintással megadhatjuk a lista megjelenési helyét. A további lépéseket az ábrák mutatják.

Lényeges, hogy az *ügyfél* táblából dolgozunk, az ügyfélazonosító értékeként annak az *id* mezőjét fogjuk felhasználni. A megfelelő ügyfél kiválasztásához ismernünk kell a nevét és települését. Lehetséges, hogy még ez sem elég, ha az adatbázisban tárolt két Farkas Boglárka ugyanabban a városban lakik. A választást megkönnyíti, ha a listában ismert szempont szerint rendezzük az ügyfeleket.

Ha az ügyfelek beviteli listáját előállítottuk, akkor az *ügyfélazonosító* beviteli mezőre már nem lesz szükségünk, törölhető.

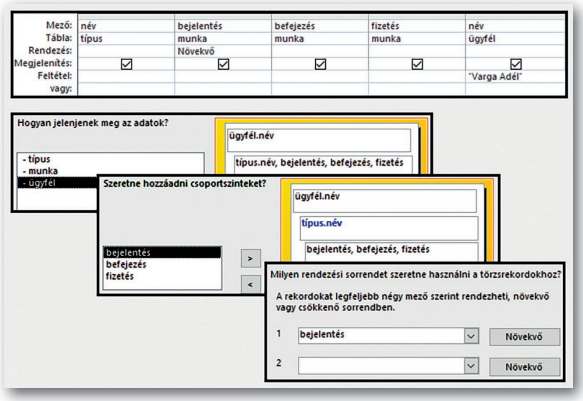
Ugyanezzel a módszerrel készíthetünk egy listát a típusazonosító helyett is, és akkor nem szükséges a munkafajták kódját ismerni az adatrögzítéshez.

36. példa: Űrlap segítségével generált jelentés

Készítsünk űrlapot, amellyel jelentést állíthatunk elő az űrlapon megadott személy által végeztetett munkákról!

Először ne is törődjünk azzal, hogy az űrlapon meg kell adnunk egy személyt! Készítsünk egy lekérdezést, amelybe írjuk be egy, az *ügyfél* táblában szereplő személy nevét, és listázzuk ki a végzett munka típusát és annak időadatait!

Az elkészült lekérdezésből készítsünk a lekérdezés minden mezőjét tartalmazó jelentést! A csoportosítást végezzük az ügyfelek, azon belül a munkatípus szerint, az adatok sorrendjét pedig a bejelentés dátuma határozza meg! Az elkészült jelentést a tanult módon formázhatjuk.

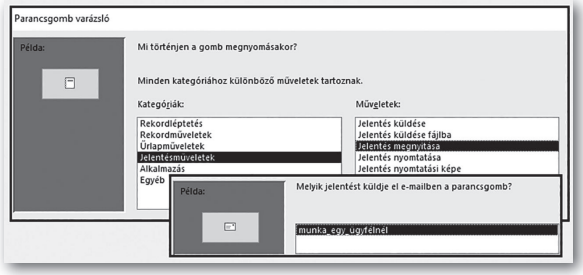


Ha a jelentést futtatjuk, akkor az automatikusan végrehajtja a lekérdezést, az abból kapott adatokból pedig elkészül a jelentés. Arra van szükségünk, hogy az űrlapról kezdeményezzük a jelentés futtatását, és a lekérdezés a feltételként szereplő nevet az űrlapból vegye át.

37. példa: Űrlap készítése *Tervező nézetben*

Készítsünk egy űrlapot *Tervező nézetben!* Az üres űrlapon helyezzünk el egy beviteli mezőt. A beviteli mező *Tulajdonságlapján* a nevét módosítsuk ügyfélnévre! Az űrlapon helyezzünk el egy gombot, amelyet az ábrán látható módon állítsunk be!

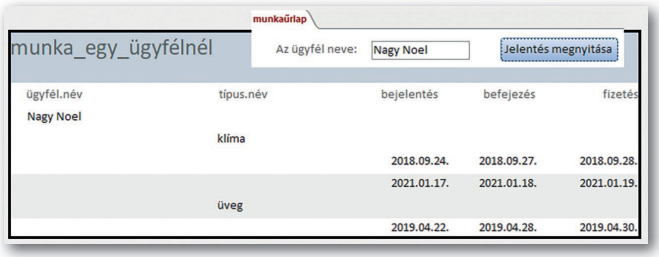
Ekkor a gombra kattintással megnyitja a képen látható jelentést. Ha kipróbáljuk, akkor még mindig nem az űrlapba írt névvel dolgozik, a futtatás eredményét az nem befolyásolja. Egyetlen feladatunk van: a lekérdezésben lecserélni a konkrét nevet az űrlap általunk elnevezett mezőjére.



Tervező nézetben töröljük ki a lekérdezésben szereplő nevet, majd a *helyi menü* > *Szerkesztés* > *Kifejezéselemek* > *javítás.accdb* > *Űrlapok* > *Betöltött űrlapok* > *munkaűrlap* > *ügyfélnév* választásával teremtsük meg a kapcsolatot az űrlappal!

Mező:	név	bejelentés	befejezés	fizetés	név
Tábla:	típus	munka	munka	munka	ügyfél
Rendezés:		Növekvő			
Megjelenítés:	☑	☑	☑	☑	☑
Feltétel:					[Űrlapok]![munkaűrlap]![ügyfélnév]
vagy:					

Ezután az űrlapba tetszőleges nevet bejegyezve megjeleníthetjük a kívánt adatokat. Fontos megemlíteni, hogy az elkészült megoldás csak mintát ad hasonló feladatok megoldásához, de nem tökéletes, hiszen ha két ügyfél neve egyezik, akkor az általuk végeztetett munkákat nem különíti el.



Feladatok

Készítsük el a következő űrlapokat az *étkezés* adatbázist használva! Ha szükséges, készítsünk lekérdezést! Az elkészített objektumokat formailag tetszés szerint tervezzük meg!

a) Készítsünk űrlapot, amely lehetőséget ad egy új ételmenü rögzítésére! Állítsuk be, hogy az *ár* csak 1000 és 2500 Ft közötti érték lehessen! *(f15a)*
b) Készítsünk űrlapot, amely egy új vendég rögzítésére alkalmas! Próbáljuk beállítani, hogy településként csak az eddig is rögzített két várost és két falut fogadja el! *(f15b)*
c) Készítsünk űrlapot, amely egy új ebédet rögzít! A vendéget és a menüt egy-egy legördülő listából válasszuk! *(f15c)*
d) Készítsünk űrlapot, amelyen szerepel egy *vezetéknév* nevű beviteli mező! Az űrlapon található gomb megnyomására fusson le egy lekérdezés, amely az űrlapba írt vezetéknevű személyek adatait listázza ki! *(f15d)*

Ami a választó lekérdezésen túl van

Ha adatbázisokkal dolgozunk, bármilyen felületen használjuk is őket, szinte mindig adatokat kérdezünk le. Egyesek feltehetik a kérdést: mi mást lehetne? Szó esett már arról, hogy az adattáblákat új rekordokkal bővíthetjük. Arról nem beszéltünk, hogy módosíthatunk az adatbázison, és törölhetünk is belőle. Ezeket a műveleteket **adatmanipulációs utasításoknak** nevezzük.

Próbáljuk ki az adatmanipulációs utasításokat a *javítás* adatbázist használva!

Törlő lekérdezés

Fontos tudni, hogy törlő lekérdezésnél általában csak egy táblából törölhetünk adatot. (Ha a kaszkádolt törlést beállítottuk a kapcsolat megadásakor, akkor azok a rekordok is eltűnnek a többi táblából, ahol az azonosító értéke idegen kulcsként szerepel.) Ha a tábla kapcsolatainál a hivatkozási integritás be van kapcsolva, akkor az a rekord nem törölhető, amelyre más táblából idegen kulcs mutat. A törlés mindig a teljes rekordot eltávolítja.

38. példa: Rekordok törlése

- Nagy Alex felsővárosi lakos tévesen került az adatbázisba. Soha egyetlen problémát nem jelentett be. Adatainak törlését kéri az *ügyfél* táblából. Készítsünk lekérdezést, amely eltávolítja az ő adatait tartalmazó rekordot! *(Alex)*
- Az adatvédelmi szabályozás miatt – bizonyos feltételek teljesülése esetén – lehet kérni a velünk kapcsolatban keletkezett adatok törlését. Töröljük a *munka* táblából a Katona Lara által rendelt munkákat! *(Lara)*
- Az *ügyfél* táblából töröljük azokat a személyeket, akiknél még nem végeztek munkát! *(nemszerepel)*

Azt javasoljuk, hogy törlő lekérdezés esetén először vegyük fel a szükséges táblákat, mintha adatlekérdezést szeretnénk készíteni. Listázzuk ki a feltételnek megfelelő rekordokat, nézzük meg az érintett rekordok körét! Ha meggyőződtünk arról, hogy valóban azok szerepelnek, amelyeket törölni akarunk, akkor állítsuk át a lekérdezés típusát törlőre, és futtassuk le! Fontos az óvatosság, mert ez a művelet nem vonható vissza.

Mező:	ügyfél.*	név
Tábla:	ügyfél	ügyfél
Törlés:	From	Where
Feltétel:		"Nagy Alex"
vagy:		

Az *Alex* lekérdezés egyetlen táblát, az *ügyfél* táblát érinti. A lekérdezőrácson a *Törlés* jelenti az újdonságot a választó lekérdezéshez képest. A FROM segítségével adjuk meg a táblát, amelyből törlünk, a WHERE segítségével pedig a feltételt!

Az *Alex* lekérdezés SQL-nyelvű megfelelője:

```
DELETE * FROM ügyfél WHERE név="Nagy Alex";
```

A *Lara* lekérdezésben már két tábla szerepel. A követendő módszer most is ugyanaz. A *munka* táblánál beállítjuk a FROM-ot, az *ügyfél* táblában fellelhető névre a WHERE-t, megadjuk a feltételt, és készen is vagyunk.

A *Lara* lekérdezés SQL-nyelvű megfelelője:

```
DELETE munka.* FROM ügyfél INNER JOIN munka ON ügyfél.id =
  munka.ügyfélid WHERE ügyfél.név="Katona Lara";
```

A *nemszerepel* lekérdezés a lekérdezőrácson nem kattintgatható ki. Az ilyen lekérdezések SQL-nyelven allekérdezést használva elkészíthetők. Ekkor a főlekérdezésben csak az a tábla szerepel, amelyből törölni szeretnénk – a feltételben a kapcsolatot megadó mezőre szűrünk.

A *nemszerepel* lekérdezés SQL-nyelvű megfelelője:

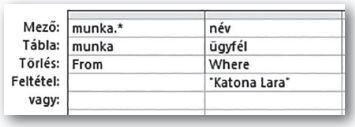
```
DELETE *
FROM ügyfél
WHERE id NOT IN (SELECT ügyfélid FROM munka);
```

Frissítő lekérdezés

Frissítő lekérdezésnél egy tábla néhány mezőjének értékét módosítjuk. Ha a módosítani kívánt mező egy másik táblában idegen kulcsként szerepel, akkor nem változtatható meg az értéke, amennyiben a kapcsolatnál a hivatkozási integritás be van kapcsolva. Ha a kaszkádolt frissítés be van állítva a kapcsolatnál, ez az érték a másik táblában is megváltozik.

39. példa: Adatok módosítása

- Egy rendelettel megszüntették a 3333-as irányítószámot, helyette a 3330-ast kell használni. Végezzük el ezt a módosítást! *(irszám)*
- Szintén a szabályozás változása miatt a *munka* táblában az adóval növelt árat kell feltüntetni az *anyagár* oszlopban a klímával kapcsolatos szereléseknél. Végezzük el ezt a módosítást! *(klíma)*

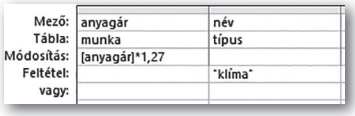


A frissítő lekérdezésnél is követhetjük a törlő lekérdezésnél leírt tanácsot: először választó lekérdezéssel ellenőrizzük az érintett rekordokat. Az ellenőrzés után alakítsuk frissítő lekérdezéssé a választót, majd futtassuk le! A megfelelő körültekintés azért fontos, mert ez a művelet sem vonható vissza.

Az *irszám* lekérdezés egyetlen táblát érint. A rácson a *Módosítás* sor jelenti az újdonságot, oda a változtatás értékét vagy képletét kell beírnunk.

Az *irszám* lekérdezés SQL-nyelvű megfelelője:

```
UPDATE ügyfél SET irszám =”3330” WHERE irszám=”3333”;
```



A *klíma* lekérdezés kéttáblás. Az anyagár mező új értékét egy képlettel tudjuk meghatározni, a korábbi érték 1,27-szorosával. Természetesen nemcsak az adott tábla, hanem akár más táblák mezőit is felhasználhatjuk a feltételben.

A *klíma* lekérdezés SQL-nyelvű megfelelője:

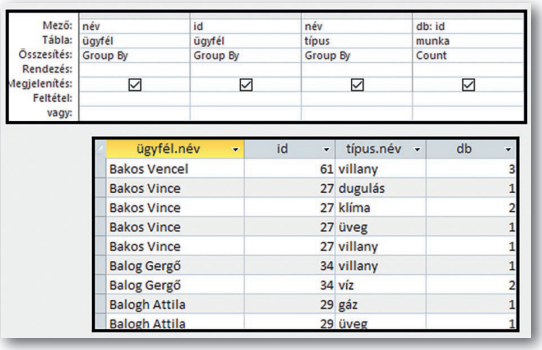
```
UPDATE típus INNER JOIN munka ON típus.id = munka.típusid SET
  munka.anyagár = [anyagár]*1.27 WHERE típus.név=”klíma”;
```

Keresztáblás lekérdezés

A keresztáblás lekérdezés nem része a klasszikus adatbázis-kezelési eszközöknek, sokkal inkább kimutatáskészítési eszköz, amely az Access program által is biztosított.

40. példa: Kétdimenziós kimutatás

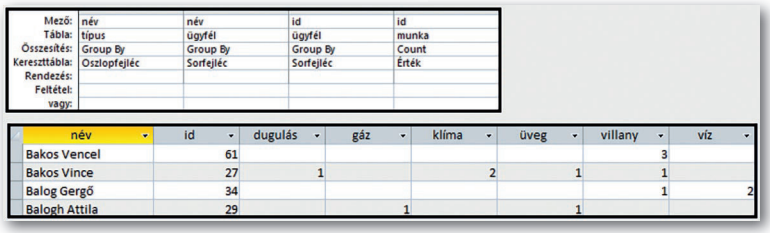
Határozzuk meg, hogy melyik ügyfél hány alkalommal végeztette el az egyes munkákat! *(stat)*



Mező:	név	id	név	db: id
Tábla:	ügyfél	ügyfél	típus	munka
Összesítés:	Group By	Group By	Group By	Count
Rendezés:				
Megjelenítés:	☑	☑	☑	☑
Feltétel:				
vagy:				

ügyfél.név	id	típus.név	db
Bakos Vencel	61	villany	3
Bakos Vince	27	dugulás	1
Bakos Vince	27	klíma	2
Bakos Vince	27	üveg	1
Bakos Vince	27	villany	1
Balog Gergő	34	villany	1
Balog Gergő	34	víz	2
Balogh Attila	29	gáz	1
Balogh Attila	29	üveg	1

Ezt a feladatot egyszerű lekérdezéssel is megoldhatjuk. Csoportosítanunk kell az ügyfelek és a munkatípusok szerint, majd megszámláltatni az elvégzett munkákat. Az ügyfelenkénti számláláshoz mindenképpen szükségünk van az ügyfél azonosítójára – vagy pontosabb címadataira –, különben az azonos nevű személyeket egyként kezelné.



Mező:	név	név	id	id
Tábla:	típus	ügyfél	ügyfél	munka
Összesítés:	Group By	Group By	Group By	Count
Kereszttábla:	Oszlopfejléc	Sorfejléc	Sorfejléc	Érték
Rendezés:				
Feltétel:				
vagy:				

név	id	dugulás	gáz	klíma	üveg	villany	víz
Bakos Vencel	61					3	
Bakos Vince	27	1		2	1	1	
Balog Gergő	34					1	2
Balogh Attila	29		1		1		

A kapott lista nehezen áttekinthető, hosszabb ideig tart meghatározni, hogy kik hányféle munkát végeztettek, melyik munkából kinél fordult elő több is.

Ennek a problémának áttekinthető megoldását kapjuk kereszttáblás lekérdezéssel. A végeredmény természetesen ugyanaz, de a meghatározott értékeket egy kétdimenziós táblázatban helyezi el. Ha az előző választó lekérdezést átalakítjuk – a lekérdezés típusát megváltoztatva – kereszttáblás lekérdezéssé, akkor a táblázat sor- és oszlopfejlécét kell megadnunk. Az adatok elemszáma alapján a sorok fejlécébe az ügyfelek neve kerüljön, az oszlopok fejlécébe pedig a munkák típusa. A lekérdezés futtatásának eredménye az ábráról leolvasható.

Feladatok

Készítsük el a következő lekérdezéseket az *étkezés* adatbázist használva!

a) Készítsünk lekérdezést, amely a menük árát 10 százalékkal megemeli! *(f16a)*
b) Egészítsük ki az *ebéd* táblát egy *eltöltött* nevű, idő típusú mezővel! Készítsünk lekérdezést, amely az *eltöltött* mező értékét kitölti az *érkezés* és *távozás* mezőben tárolt adatok alapján! *(f16b)*
c) Készítsünk lekérdezést, amely a nagyfalui Kőrös utcát Tisza utcára cseréli! *(f16c)*
d) Készítsünk lekérdezést, amely kitörli a Fekete Kamilla által fogyasztott ebédek adatait! *(f16d)*
e) Készítsünk lekérdezést, amely kitörli Fekete Kamilla adatait! (f16e)
f) Készítsünk kereszttáblás lekérdezést, amely megmutatja, hogy az egyes vendégek külön-külön hányszor választották az egyes menüket! *(f16f)*

Vegyes feladatok

Készítsük el a következő lekérdezéseket a *város* adatbázist használva!

a) Soroljuk fel ábécérendben a Heves megyei városokat! *(f17a)*
b) Listázzuk ki azokat a városokat, melyek nevében van *e* betű! *(f17b)*
c) Adjuk meg azokat a városokat, melyek nevében nincs *e* betű! *(f17c)*
d) Határozzuk meg azokat a városokat, melyek nevében a magánhangzók közül csak *e* betű van! *(f17d)*
e) Adjuk meg, melyik megyében található a legkisebb területű város! *(f17e)*
f) Listázzuk ki azokat a megyei jogú városokat, amelyek nem megyeszékhelyek! *(f17f)*
g) Határozzuk meg, hogy hány város népesebb 75 ezer főnél! *(f17g)*
h) Adjuk meg, hogy hány fő lakik a legkisebb és a legnagyobb lélekszámú Fejér megyei városokban! *(f17h)*
i) Készítsünk lekérdezést, amely megadja a megyeszékhelyek átlagos lélekszámát! *(f17i)*
j) Adjuk meg azokat a megyéket, amelyekben több megyei jogú város is található! *(f17j)*
k) Listázzuk ki az Egernél népesebb városokat! *(f17k)*
l) Melyik megyében van ugyanannyi város, mint Hevesben? *(f17l)*
m) Soroljuk fel annak a megyének a városait, amelyben a legkevesebb város található! *(f17m)*

Mesterséges intelligencia

A gondolkodó gép megalkotása régóta foglalkoztatja az emberiséget. Az első ilyen informatikai eszközöket az 1950-es években alkották, és akkor kapta a terület a **mesterséges intelligencia** (MI vagy az angol Artificial Intelligence kifejezésből az AI) elnevezést. Gyors fejlődésnek azonban csak az elmúlt évtizedben indult. A mesterséges intelligencia fejlődése összekapcsolódik az információs társadalom újonnan kialakuló lehetőségeivel, és kétségkívül maga is jelentősen hozzájárul az információs társadalom jövőjének alakulásához.

Mesterséges intelligencia alatt olyan gépet, rendszert értünk, amely képes az emberi viselkedés, gondolkodás utánzására, problémamegoldásra. Fontos tudnunk, hogy a mesterséges intelligencia egy-egy részterületen képes az emberi intelligencia leképezésére, akár jobban is teljesíthet bizonyos jól meghatározott feladatokat.

Egyelőre azonban igen messze áll attól, hogy az emberi gondolkodást teljes komplexitásában helyettesíteni tudja vagy felülmúlja.

A terület mai rohamos fejlődéséhez több tényező járul hozzá. Egyik fontos tényező a számítási kapacitás növekedése, ugyanis a mesterséges intelligencia által használt algoritmusok általában műveletigényesek. Sok olyan chipet gyártanak, amely támogatja a mesterséges intelligenciát. A mobiltelefonjaink egy részében ilyen processzort találhatunk. A mesterséges intelligencia egyik fontos működési elve azon alapszik, hogy a rendszer nagy mennyiségű adat megvizsgálásával tapasztalatokat szerez, ezeket elemezve, feldolgozva, a szabályszerűségeket megfigyelve alakítja ki a helyzetekre adandó megfelelő válaszokat. Ezt a folyamatot nevezzük **gépi tanulásnak.**

Az információs társadalomban óriási mennyiségű adatot állítunk elő. A web 2.0 alkalmazások használatakor a felhasználók saját maguk által készített tartalmakat osztanak meg. Egyre több tevékenységünket rögzítik használati eszközeink (például tartózkodási hely, meglátogatott oldalak, elkészített fényképek, online vásárlások, keresések). A digitalizáció következtében a cégek rengeteg, a tevékenységükhöz kapcsolódó adathoz jutnak

▸ Gépi tanulás

hozzá (gyártási, forgalmi adatok, vásárlók adatai stb.). Egyre több eszközünk kapcsolódik a hálózathoz. Köztük van számos olyan, amelynek hagyományosan nem ez az alapfunkciója, de a hálózati kapcsolattal többletszolgáltatást nyújthat számunkra. Ezek képesek a hálózaton keresztül kommunikálni más eszközökkel, adataikat megosztani és ezt felhasználni az optimális működésük érdekében. Ezek az eszközök alkotják a **dolgok internetét (Internet of Things: IoT)**.

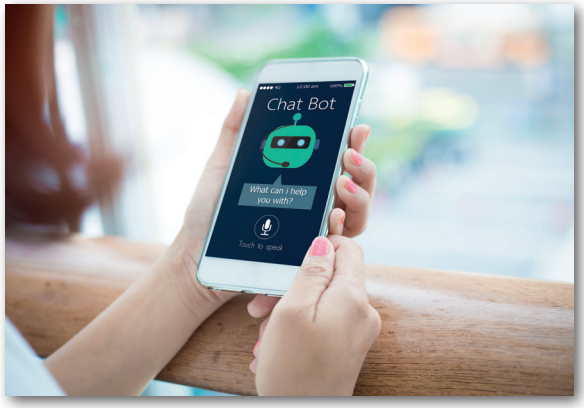
Az IoT a napról napra gyarapodó adatmennyiség elemzésére, feldolgozására új lehetőségeket nyit a gazdaság, a tudomány és a mindennapi élet számos területén. Ez az adatmennyiség nagyságrendileg nagyobb, mint amennyit korábban kezeltünk. Az adatok gyűjtése az IoT-eszközöknek, a hálózati kommunikációnak köszönhetően jelentősen felgyorsult. Feldolgozásuk, elemzésük új eljárásokat igényel. Az ezek mentén kialakuló terület a **Big Data**.

Bár gyakran észre sem vesszük, életünkben már ma is fontos szerepet kapnak a mesterséges intelligencia által működtetett eszközök, megoldások, és egyre nagyobb mértékben járulnak hozzá a fejlődéshez.

A mesterséges intelligencia alkalmazása napjainkban

Lássunk néhány általánosan használt alkalmazást, amelyek mögött mesterséges intelligencia áll.

- Az online kereséskor használt **keresőmotorokat** mesterséges intelligencia működteti.
- Az idegen nyelven megjelenő online tartalmak **automatikus fordításakor** is ezt használjuk.
- Alkalmas a beszéd, a zene és a kép felismerésére. Ez segíti a képalapú keresést, azokat az alkalmazásokat, amelyekkel fel tudjuk ismertetni a hallott zeneművet. Ezt használjuk, amikor hanggal irányítunk egy programot, de a videók automatikus feliratozását is ez végzi.
- Egyre több ügyfélszolgálat, weboldal használ **chatbotot,** azaz beszélgető robotot. Ezek a legygyakoribb kommunikációs helyzetekben képesek az ügyféllel beszélgetést folytatni és az egyszerűbb ügyeket megoldani. Ezzel enyhítenek az ügyfélszolgálat leterheltségén, gyorsabb kiszolgálást tesznek lehetővé. A lefolytatott párbeszédek adataiból tapasztalatokat gyűjtenek, így egyre jobb reakciót képesek adni a kialakuló helyzetekre.



► Chatbot

- A közösségi oldalak személyre szabott tartalma a mesterséges intelligencia segítségével készül. A viselkedésünkre vonatkozó megfigyelések alapján jelenik meg számunkra a várhatóan legkedveltebb tartalom, a személyre szabott reklám. Hasonló módszer alapján képesek személyre szólóan filmeket, zeneszámokat ajánlani a népszerű streamingszolgáltatók.

- **Biometrikus azonosítást,** ujjlenyomat-olvasót, arcfelismerő rendszert számos helyen használunk. Ezek is a mesterséges intelligencia segítségével működnek.
- Találkozhatunk már **önvezető járművekkel.** Ezeket ma még inkább az egyszerűbb forgalmi helyzetekben alkalmazzák. Ilyen például a budapesti 4-es metró, amelyet mesterséges intelligencia vezet.
- Idetartoznak a különböző célfeladatokat ellátó **robotok,** például a robotporszívók, a gyártási folyamatokat elvégző robotok.
- A számítógépes játékok gyakran használnak mesterséges intelligenciát. Ezzel élvezetesebbé teszik a játékot, azt az érzetet keltik, hogy egy valódi partnerrel versenyzünk.

Fejlődési irányok, társadalmi hasznosság

A mesterséges intelligencia fejlődése egyre gyorsuló tendenciájú. Egyre több az olyan terület, amelynek meghatározza a fejlődését. Komoly eredményeket érhetünk el használatával a gazdaság, a tudomány, a mindennapi életünk számos területén. Segíthet a folyamatok optimalizálásában, például a forgalomszervezésben, a jobb energiafelhasználásban, a gyártási folyamatok hatékonyabbá tételében. Pontosabbá és gyorsabbá teheti a diagnosztizálást, legyen szó betegségekről vagy gépek meghibásodásáról. A nagy mennyiségű adat megvizsgálásával képes előre jelezni olyan problémákat, amelyeket nélküle nehezebben ismerhetnénk fel. Használhatjuk ezért egészségmegőrzésre, bűnmegelőzésre, katasztrófavédelemre, termeléskiesés megelőzésére. Várhatóan átvesz tőlünk munkafolyamatokat, de egész munkaköröket is képes lehet ellátni. Alkalmas lehet arra is, hogy új tartalmat állítson elő – mesterséges intelligencia segítségével lehet például zenét komponálni, irodalmi műveket létrehozni. Képes olyan emberi arcokat előállítani, amelyek teljesen valósnak tűnnek, de nem élő emberekhez tartoznak. Tanulási folyamatuk lényegesen gyorsabb, mint az embereké. Az egyik eszköz által begyűjtött ismeretek a többi hasonló feladatot ellátó eszközzel azonnal megoszthatók, vagyis azonnal használhatják a másik eszköz által megszerzett összes ismeretet.

▸ Önvezető autók

Látható, hogy a mesterséges intelligencia számos területen segítheti az emberiség életét, a fejlődést, de nagyon fontos a megfelelő szabályozása. Csak kritikus gondolkodás mellett érdemes felhasználnunk, mert **alkalmazása veszélyeket is rejthet magában.** Társadalmi, erkölcsi és technikai problémát is okozhat, ha nem megfelelően kezeljük. Például a munkaerő kiváltására használt mesterséges intelligencia adhat nekünk rövidebb munkaidőt, kényelmesebb munkavégzést, ha megfelelően alkalmazzuk, de hozzájárulhat dolgozók munkanélkülivé válásához is. Az adatok széles körű felhasználása alkalmas a technika további fejlesztésére, de a hamis vagy manipulált adatok ezt az irányt eltéríthetik. Fontos, hogy az adatok anonimizálásáról megfelelően gondoskodjunk, hogy betartsuk az adatvédelmi szabályokat. Gyakran felmerül a mesterséges intelligenciával működő eszközök esetében a **felelősség kérdése.** Ki lesz a hibás abban az esetben, ha a mesterséges intelligencia nem a megfelelő döntést hozza, és ezzel valamilyen, akár nagyobb bajt okoz?

Azért lényeges ismerni az információs társadalom ezen területét, hogy képesek legyünk megfelelően szabályozni és kézben tartani. Ebben a tekintetben az állampolgároknak, a cégeknek és az államnak is lesz felelősségük.

Kérdések, feladatok

1. Keressük meg a mesterséges intelligenciával működő *Blob Opera* alkalmazást, amelyet operaénekesek segítségével tanítottak meg énekelni! Próbáljuk ki, alkossunk zenét vele!
2. Nézzünk utána, miről nevezetes a Google *DeepMind* algoritmusa!
3. Keressünk példát olyan mesterségesintelligencia-alkalmazásokra, amelyekkel találkoztunk már (ismert chatbotok, járművek, robotok stb.)! Beszéljük meg közösen a talált lehetőségeket! Ha szükséges, nézzünk utána!
4. Nézzünk utána, mi az a *deepfake!* Miért veszélyes?
5. Milyen veszélyes helyzeteket tudunk elképzelni, amelyeket a mesterséges intelligencia alkalmazása okozhat? Hogyan lehet ezeket kivédeni?
6. Milyen konkrét eszközök tartozhatnak az *IoT* kategóriába? Keressünk példákat arra, hogy hogyan egészülhetnek ki az eszköz alapszolgáltatásai! Mi magunk milyen IoT-eszközöket használunk?
7. Mit jelent az okosotthon kifejezés? Milyen szolgáltatásokat nyújt használójának?

Kriptográfiai alapfogalmak

A napi számítógép-használat során gyakran találkozunk az operációs rendszer és a hálózati összetevők (eszközök, protokollok) biztonsági mechanizmusaival. Adataink védelme érdekében alapvető fontosságú ezek megismerése és megértése. Minden operációs rendszer biztosítja számunkra a lokális gépen, illetve lokális hálózaton tárolt adatfájljaink hozzáférés-védelmét, amely a felhasználó azonosításán és hitelesítésén alapul (DAC: discretionary access control). Ez azt jelenti, hogy csak a felhasználónév (azonosítás) és a hozzá tartozó jelszó (hitelesítés) megadásával történt bejelentkezés után férhetünk hozzá az adatfájlokhoz (és egyéb védendő objektumokhoz), és csak olyan módon, ahogyan az az egyes fájlokon/objektumokon engedélyezett számunkra (például csak olvasás, de írás nem).

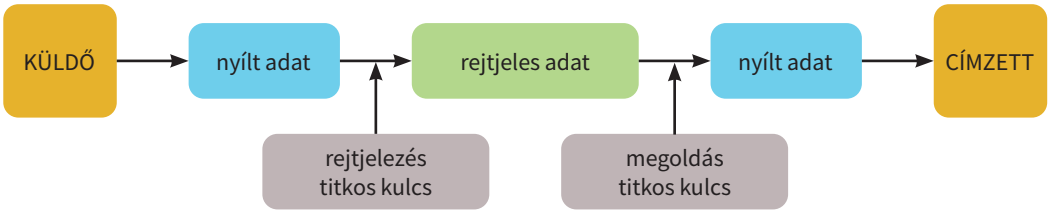
Ez a fajta védelem nem terjed ki az interneten elérhető adatokra és erőforrásokra. Ezért az internetről letöltött/feltöltött adatok hitelességét és hozzáférés-védelmét más módon kell biztosítani. A továbbiakban ezeket a biztonsági mechanizmusokat tekintjük át, amelyek megértéséhez néhány egyszerű kriptográfiai fogalmat kell tisztázni.

Rejtjelezés

Az adatok védelmének érdekében rejtjelezést az ősidők óta használnak. Célja, hogy a nyilvános csatornán (például internet) továbbított bizalmas adatokat védjük az illetéktelen megismerés ellen. Ezért a **nyílt adatot** csak a küldő és a címzett által ismert **kulccsal** rejtjelezzük, majd a **rejtjeles adatot** küldjük át a nyilvános (védtelen) csatornán. A címzett a közös kulcs ismeretében **meg tudja oldani** a rejtjeles adatot, így olvashatóvá válik számára a nyílt adat. A rejtjelezési eljárást **rejtjelalgoritmusnak** nevezzük. Nagyon egyszerű rejtjelalgoritmus lehet például az, hogy a nyílt szöveg minden betűjét más betűre cseréljük, így téve értelmezhetetlenné a rejtjeles szöveget. Ebben az esetben a kulcs a betűcseréket leíró permutáció. Természetesen ez az algoritmus szinte semmit sem ér, mivel egy kicsit is intelligensebb kódfejtő a kulcs ismerete nélkül is képes visszanyerni a rejtjeles szövegből a nyílt szöveget. Ezt a műveletet nevezzük **megfejtésnek** (vagy rejtjelfejtésnek). Összefoglalva a most megismert fogalmakat:

- **Nyílt adat:** a rejtjelezendő adat.
- **Rejtjeles adat:** a nyílt adat rejtjelezett formája, amely nyilvános csatornán továbbítható.
- **Rejtjelalgoritmus:** a rejtjelezés műveletét leíró algoritmus.
- **Kulcs:** a rejtjelalgoritmus működését meghatározó titkos adat (csak a küldő és a címzett ismerheti).
- **Rejtjelezés:** rejtjeles adat előállítása a nyílt adatból a kulcs ismeretében, rejtjelalgoritmus használatával.
- **Megoldás:** nyílt adat előállítása a rejtjeles adatból a kulcs ismeretében, a rejtjelalgoritmus használatával.
- **Megfejtés:** nyílt adat előállítása a rejtjeles adatból, a kulcs ismerete nélkül. A rejtjelalgoritmus vagy ismert, vagy nem (nehezebb eset).

A leírt rejtjelezési sémát **szimmetrikus kulcsú rejtjelezésnek** nevezzük, mivel a rejtjelezéshez és a megoldáshoz is ugyanazt a kulcsot használjuk.



▶ Szimmetrikus kulcsú rejtjelezés

Léteznek **aszimmetrikus kulcsú rejtjelezési algoritmusok** is, ahol a rejtjelező- és a megoldókulcs különböző. A példaként leírt betűcserés algoritmusnál (szaknyelven: egyszerű helyettesítés) jóval fejlettebb algoritmusokat használunk. Közkeletű tévedés, hogy megfelelő erőforrás-ráfordítással minden rejtjelezés feltörhető (megfejthető). Ez egyrészt elvileg sem igaz (mivel léteznek elvileg sem megfejthető rejtjelezések), másrészt a modern algoritmusok gyakorlatilag csak a kulcs teljes kipróbálásával fejthetők meg. A teljes kipróbálás azt jelenti, hogy az összes lehetséges kulcsot végig kell próbálni a rejtjelalgoritmussal, míg meg nem kapjuk a nyílt adatot. Ha például a kulcs 128 bites, akkor 2^{128} számú esetet kell tekintetbe venni a megfejtés során (128 bit körülbelül 20-21 beírható karakternek felel meg). A modern algoritmusok esetén (például AES, IDEA, HC128) a mai tudásunk szerint ez gyakorlatilag kivitelezhetetlen. Ez nem jelenti azt, hogy ne lenne támadható a rejtjelezés, ha gyenge kulcsot adtunk meg.

Gyenge (begépelt) kulcsok ellen jól ismert módszer a **szótáralapú támadás.** Ekkor egy szótárból veszik a kipróbálandó jelszavakat vagy azok némileg módosított formáit, annak reményében, hogy a rejtjelező pont azt a szót választotta kulcsként.

Az interneten használt biztonságos protokollok (például https) is alkalmaznak szimmetrikus kulcsú rejtjelezést, azonban nyilván nem begépelt, hanem véletlenszerűen generált kulccsal. Itt a fő kérdés az, hogy hogyan alakítanak ki közös szimmetrikus kulcsot a kommunikáló felek. Erre a problémára az aszimmetrikus kulcsú rejtjelezés nyújt megoldást.

Hitelesítés

A hitelesítés célja, hogy az adat és/vagy a forrás hitelességét bizonyítsa. Az adat hitelessége azt jelenti, hogy a kommunikáció során az adat nem sérült, vagyis valóban a feladó által eredetileg elküldött adatot kaptuk meg sértetlen állapotban. Az átviteli hibákból adódó (tehát nem rosszindulatú támadás miatti) adatsérülés különböző kontrollösszegek (például CRC – Cyclic Redundancy Check) alkalmazásával észlelhető és javítható. Rosszindulatú támadás ellen ez nem véd, mivel a támadó a manipulált adatra számolhat helyes kontrollösszeget. Az ilyen támadás ellen a **digitális aláírás** védi az adatot, amely az aláíró személy azonosságát (a forrás hitelességét) is bizonyítja. Tehát a hagyományos, papíralapú dokumentum aláírásával szemben a digitális aláírás nemcsak az aláíró személyét bizonyítja, hanem az aláírt dokumentum tartalmát és akár idejét is (vagyis azt, hogy valóban az aláíró személy, valóban azt a tartalmú dokumentumot és valóban abban az időpontban írta alá).

A **digitálisan aláírt** (vagy másként: elektronikusan hitelesített) **dokumentum** aláírását természetesen csak számítógépen lehet ellenőrizni. Vagyis a papírra kinyomtatott, elektronikusan hitelesített dokumentum **nem hiteles,** hiába látható rajta az „elektronikusan

hitelesített” pecsét. Tipikus példa erre a Földhivataltól kikért, elektronikusan hitelesített, kinyomtatott tulajdoni lap, amelyet **nem szabad hitelesnek elfogadni,** hiszen az „elektronikusan hitelesített” pecsétet bárki rászerkesztheti bármilyen papíralapú dokumentumra. A digitálisan aláírt PDF-fájlt megfelelő PDF-olvasó programmal (Acrobat Reader, Foxit Reader stb.) betöltve lehet (és kell) ellenőrizni. Ez hibát jelez, ha a dokumentum tartalma megváltozott az eredeti (aláírt) tartalomhoz/időponthoz képest, vagy ha nem a deklarált személy/hivatal írta alá.

Digitális aláírást egyaránt lehet alkalmazni nyílt (például tulajdoni lap) és rejtjeles dokumentumon, illetve adatfolyamon. Ritkább eset, hogy a dokumentumot/adatfolyamot csak rejtjelezik, de nem hitelesítik, bár maga a rejtjelezés is biztosít bizonyos szintű (sokszor elegendő) adat- és forráshitelességet. Ugyanis ha a címzett a közös kulccsal meg tudja oldani a rejtjeles adatot, és értelmes szöveget kap vissza, továbbá biztos benne, hogy a közös kulcsot csak a kommunikáló felek ismerik, akkor ez önmagában is bizonyítja az adat és a forrás hitelességét. Az interneten használt biztonságos protokollok azonban mindig alkalmaznak hitelesítést is. Manapság megfigyelhető, hogy alig akad olyan nyilvánosan elérhető weboldal, amely nem a biztonságos https-protokollt, hanem a nyílt http-protokollt használja. Ennek oka elsősorban az adat és forrás hitelességének bizonyítása, és nem a tartalom rejtjelezése (hiszen az nyilvános).

Kérdések, feladatok

1. Nézzünk utána, mit jelent a Caesar-kód vagy Caesar-rejtjel! Miért fejthető meg könnyen az így kódolt üzenet?
2. Mi a különbség a rejtjelezett adat megoldása és megfejtése között?
3. Milyen nyelvi jellemzők alapján ismerhető fel az egyszerű helyettesítés? Hogyan lehetne megfejteni?
4. Nézzünk utána, mi az az Enigma! Hogyan kapcsolódik a történelemi eseményekhez és a számítógépekhez?
5. Készítsünk programot, amely egy szöveg titkosítását valósítja meg egyszerű helyettesítés segítségével!

Aszimmetrikus kulcsú titkosítás

Hashfüggvény

A digitális aláírás működésének megértéséhez még meg kell ismerkednünk a hash- (hasító) függvény fogalmával.

> A hashfüggvény olyan matematikai művelet, amely tetszőleges hosszúságú adatfolyamhoz fix hosszúságú bitsorozatot rendel.

Az adatfolyam **hashértéke** tulajdonképpen az adatfolyamra jellemző kivonat. Ez azt jelenti, hogy ha az adatfolyam egyetlen bitje megváltozik, akkor a hashértéke teljesen más lesz (a hashérték bitjeinek körülbelül a fele megváltozik). Vagyis gyakorlatilag lehetetlen egy adott hashértékhez másik adatfolyamot találni, amelynek ugyanez lesz a hashértéke. Természetesen végtelen sok olyan adatfolyam létezik, amelynek ugyanaz a hashértéke, hiszen a fix hosszúságú hashértékek száma véges (például ha a hash 160 bites, akkor 2^{160} darab különböző hashérték van), míg az adatfolyamok száma végtelen.

Két azonos hashértékű adatfolyam esetén **hashütközésről** beszélünk. A hash elleni támadás során az adott hashértékhez kell találni olyan adatfolyamot, amelynek éppen ez az érték a hashértéke. Ez kriptográfiai hash- (SHA1-, MD5-) függvények esetén gyakorlatilag kivitelezhetetlen (bár az MD5 128 bites hasht már nem tekintik kriptográfiailag biztonságosnak). Az átviteli hibák ellen védő kontrollösszegek (például CRC) szintén hashfüggvénynek tekinthetők, de semmiképpen nem kriptográfiailag biztonságosnak.

Hashfüggvény képezhető rejtjelező algoritmusból is, például úgy, hogy a hashelendő adatfolyamból kulcsot képeznek, amellyel fix nyílt adatot (például a hash hosszúságának megfelelő csupa 0 bitet) lerejtjelezik, és ezt tekintik hashértéknek. Tehát itt a nyílt adat (a hash hosszúságának megfelelő csupa 0 bit) és annak rejtjeles képe ismert (ez a hashérték), ebből azonban a kulcsot (a hashelt adatfolyamot) nem lehet visszaállítani.

A **digitális aláírás folyamata** a következőképpen történik: az aláírandó dokumentumnak kiszámolják a kriptográfiai hashértékét (például 160 bites SHA1), amelyet az aláíró kulcsával lerejtjeleznek, és mellékelnek a dokumentumhoz. Az aláírást a másik fél ellenőrizni tudja, ha ismeri az aláírókulcsot, ugyanis ugyanezt a műveletet megismételve ugyanezt az eredményt kell kapnia (pre-shared key alapú digitális aláírás). A nagy probléma ezzel a módszerrel az, hogy bárki produkálhat megfelelő aláírást, aki ellenőrizni tudja, hiszen ismeri az aláírókulcsot. Bizonyos protokollok ennek ellenére használják a pre-shared key alapú digitális aláírást (például az IPsec gatewayek). A fenti probléma az aszimmetrikus kulcsú rejtjelezéssel kezelhető, amikor a rejtjelező- és megoldókulcs különböző.

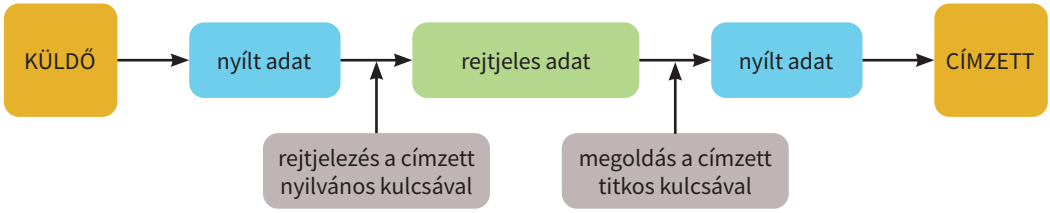
Aszimmetrikus kulcsú rejtjelezés

Az interneten zajló kommunikáció során közvetlenül nem alkalmazható a szimmetrikus kulcsú rejtjelezés és hitelesítés (aláírás), hiszen a kommunikáló felek a nyílt csatornán nem küldhetik át a szimmetrikus kulcsaikat. Ezért más megoldást kell találni. Az alapötlet az, hogy mindegyik fél saját kulcspárral rendelkezik, amelyikből az egyik **titkos** (private) **kulcs,** a másik pedig a **nyilvános** (public) **kulcs.** A titkos kulccsal rejtjelezett adat a neki megfelelő

nyilvános kulccsal oldható meg (hitelesítés, aláírás). Azonban a nyilvános kulcs is használható rejtjelezésre, amely a neki megfelelő titkos kulccsal oldható meg (nyilvános kulcsú rejtjelezés). Ez úgy képzelhető el, mint egy lakat, amelyhez két különböző kulcs tartozik. Az egyik kulccsal bezárt lakatot csak a másik kulccsal lehet kinyitni, azonban bármelyik kulccsal zárható a lakat. A két kulcs összetartozik: a titkos kulcsnak csak az adott nyilvános kulcs lehet a párja, és viszont. Az interneten kommunikáló felek aszimmetrikus kulcsok segítségével, a következőképpen kommunikálnak: mindegyik fél generál saját maga számára egy aszimmetrikus kulcspárt. A kulcspár egyik kulcsát kinevezi titkos kulcsnak, amelyet ezután titokban tart, és nem ad ki senkinek. A kulcspár másik kulcsát pedig kinevezi nyilvános kulcsnak, és közzéteszi. Bárkinek elküldi, aki kommunikációt kíván kezdeményezni vele. Az aszimmetrikus kulcspár rejtjelezésre és aláírásra egyaránt használható.

Rejtjelezés aszimmetrikus kulccsal

Tegyük fel, hogy András rejtjeles üzenetet kíván küldeni Beának. Nincs még közös szimmetrikus kulcsuk, ezért András elkéri Bea nyilvános kulcsát. Mivel ez nyilvános kulcs, küldhető nyílt csatornán. András Bea nyilvános kulcsával rejtjelezi az üzenetet, amelyet Bea a saját titkos kulcsával meg tud oldani. A gyakorlatban az aszimmetrikus kulcsú rejtjelezést szimmetrikus kulcs kicserélésére használják (például PGP), vagyis András üzenete a javasolt szimmetrikus kulcs lesz, mivel a szimmetrikus kulcsú rejtjelezés sokkal gyorsabb.



▶ Rejtjelezés aszimmetrikus kulccsal

Aláírás aszimmetrikus kulccsal

Tegyük fel, hogy András olyan üzenetet kíván küldeni Beának, amelyben bizonyítja, hogy valóban ő az üzenet írója, és valóban az az üzenet tartalma, amit aláírt. Az üzenet nem feltétlenül rejtjelezett. Először is, András elküldi saját nyilvános kulcsát Beának, aki majd ezzel tudja ellenőrizni András aláírását. András megírja az üzenetet, hashértéket számol rá (például SHA1-et), majd a hashértéket rejtjelezi a saját titkos kulcsával. Ez a rejtjelezett hashérték lesz az üzenet aláírása, amelyet az üzenethez mellékelve szintén átküld Beának. Bea megoldja az aláírást András nyilvános kulcsával, és az eredményt összeveti az általa szintén kiszámolt hashértékkel. Ha a két érték egyezik, akkor az üzenet tartalma sértetlen, és valóban András az aláíró.

Mint a fenti példákból megfigyelhető, rejtjelezésnél a partner nyilvános kulcsával rejtjelezzük az üzenetet, amit ő a titkos kulcsával tud megoldani. Aláírásnál a hashértéket saját titkos kulcsunkkal rejtjelezzük, amit a partner a mi nyilvános kulcsunkkal tud ellenőrizni. Tehát a rejtjelezésnél a nyilvános kulcsot rejtjelezésre, a titkos kulcsot megoldásra használjuk, míg aláírásnál a titkos kulcsot rejtjelezésre, a nyilvános kulcsot pedig megoldásra használjuk.

► Aláírás aszimmetrikus kulccsal

A legismertebb és legszélesebb körben használatos aszimmetrikus kulcsú rejtjelrendszer az RSA (Rivest–Shamir–Adleman). RSA-kulcspár generálásánál két nagyon nagy prímszámot (1024–4096 bitest) választanak, amelyek szorzatából képezik a nyilvános kulcsot (nem pont a szorzat lesz a nyilvános kulcs!). Hogy a nyilvános kulcsból a hozzá tartozó titkos kulcsot ki lehessen számolni, ezt a szorzatot kellene tényezőkre bontani. Ez azonban ekkora prímek esetén gyakorlatilag megoldhatatlan feladat. Ez adja az **RSA-módszer** erejét. Ezenkívül léteznek más aszimmetrikus kulcsú rendszerek is.

A most vázolt aszimmetrikus kulcsú rejtjelezéssel és aláírással van egy nagyon nagy probléma: a nyilvános kulcsok közzététele/terjesztése nem hitelesített. Ez azt jelenti, hogy András ugyan elkéri Bea nyilvános kulcsát (és Bea Andrásét), de nem lehet biztos benne, hogy valóban Bea nyilvános kulcsát kapta meg. Egy közbeékelődő támadó (man-in-the-middle attack) eljátszhatja Bea szerepét András felé, és András szerepét Bea felé. Ekkor mindketten a támadó nyilvános kulcsát kapják meg, azt gondolván, hogy az a partnerük nyilvános kulcsa. A támadó olvashatja a rejtjeles üzeneteket, átírhatja a tartalmukat, és hamisíthatja az aláírásokat. Mindebből András és Bea semmit nem vesznek észre. Ennek a problémának a megoldására jöttek létre a **tanúsítványok (certificate)** és a **tanúsítványkibocsátó szervezetek** (certificate authority: CA).

Tanúsítványok

A tanúsítványok – nagyon leegyszerűsítve – tanúsítványkibocsátó szervezet által digitálisan aláírt nyilvános kulcsok. Emellett számos egyéb információt is tartalmaznak (a tulajdonos neve, https esetén doménnév, lejárati dátum stb.). Tanúsítványok alkalmazásával kivédhetők a közbeékelődő támadások, mert a CA garantálja, hogy az általa aláírt tanúsítvány valóban a tanúsítványban megnevezett entitáshoz (magánszemély, szervezet, szerver) tartozik (https-tanúsítványban a doménnév is szerepel). A tanúsítvány ellenőrzése az aláíró CA nyilvános kulcsa (tanúsítványa) segítségével történik, ami az ellenőrzést végző gépen már eleve el lett tárolva (például a Windows telepítésekor). Tehát az egész infrastruktúra működése azon alapul, hogy a kommunikáló felek megbíznak egy harmadik félben (ez a CA), aki biztosítja őket a másik fél személyazonosságáról. Ezt a modellt **„Trusted-Third-Party”-modellnek** nevezik (röviden: TTP). A kommunikáló felek nem tudják ellenőrizni, hogy a harmadik fél az ő érdekükben jár el (például nem ad ki hamis tanúsítványt), ezért kénytelenek megbízni benne, hasonlóan ahhoz, mint ahogy az állami hivatalok tisztességes működésében is megbízunk. Ez a fajta bizalmi viszony a harmadik fél (CA) felé akkor jön létre, amikor például a Windows operációs rendszert telepítjük a gépünkre, melynek során a megbízhatónak nyilvánított CA-k tanúsítványa bekerül az operációs rendszer tanúsítványtárába. Vagyis, ha megbíztunk az operációs rendszer telepítőcsomagjának eredetiségében, akkor

a feltelepített CA-tanúsítványokban (és ezzel együtt magukban a CA-kban) is meg kell bíznunk. A feltelepített CA-tanúsítványok között az összes, úgynevezett root CA- (gyökér CA-) tanúsítvány megtalálható lesz (lásd később).

A tanúsítványkibocsátó szervezetek (CA) adott esetben kiadhatnak alárendelt tanúsítványkibocsátó szervezetek (röviden: al-CA) részére olyan tanúsítványt, amellyel azok szintén kiadhatnak tanúsítványokat. Így egy **tanúsítványlánc** jön létre. A tanúsítványlánc tetején a root CA áll. A root CA-tanúsítványát értelemszerűen csak saját maga képes aláírni, ezért az ilyen tanúsítványt **önaláírtnak** nevezzük. Csak root CA esetén tekinthető érvényesnek az önaláírt tanúsítvány. A tanúsítványlánc alján a **végfelhasználói tanúsítvány** áll. Végfelhasználói tanúsítvánnyal már nem lehet újabb tanúsítványt aláírni (kibocsátani). A tanúsítványláncban szereplő CA-k nem rendelkeznek az általuk kibocsátott tanúsítványok nyilvános kulcsának titkos párjával.

A különböző szintű tanúsítványok a szintjüknek megfelelő adminisztratív eszközökkel igényelhetők. Például nagyvállalati szintű CA-tanúsítványt a megfelelő országos szintű CA-tól kell igényelni. Az igénylés során adminisztratív módon (céges papírok, közjegyző, személyes megjelenés, személyazonosítás stb.) kell bizonyítani az igénylő entitás személyazonosságát és jogosultságát, aminek jelentős anyagi vonzata is lesz. A végfelhasználói (tanúsítványkibocsátásra alkalmatlan) tanúsítvány igénylése lényegesen egyszerűbb és olcsóbb. Nincs feltétlenül szükség adminisztratív személyazonosításra, és akár ingyen is igényelhető (például Let's Encrypt). Természetesen ebben az esetben is szükség van valamiféle bizonyítékra, hogy például (https esetén) az adott domén a mi felügyeletünk alá tartozik.

A tanúsítványlánc ellenőrzése a végfelhasználói tanúsítványtól indul. A láncban felfelé haladva a kibocsátó CA nyilvános kulcsával ellenőrizzük a tanúsítványkibocsátó általi aláírást, míg el nem jutunk a root CA-hoz. A root CA önaláírt tanúsítványa természetesen érvényes, mivel szerepel az operációs rendszer tanúsítványtárában. Ha az ellenőrzési lánc valahol megszakad, akkor a végfelhasználói tanúsítvány érvénytelen.

András és Bea kommunikációjában a tanúsítványok használata a következőképpen néz ki: mindketten RSA-kulcspárt generálnak maguk számára, majd a nyilvános kulcsukat és egyéb adatokat átadják egy megbízható CA-nak. A CA, miután meggyőződött az igénylő személyazonosságáról, a kapott adatokból előállítja a tanúsítványnak megfelelő adatstruktúrát, amelyet a saját titkos kulcsával aláír. Az így létrejött adatstruktúrát nevezzük tanúsítványnak. Ezután András és Bea között a már ismertetett módon zajlik a kommunikáció. Mindkettőjüknek a másik nyilvános kulcsára van szüksége (akár rejtjelezésről/kulcscseréről, akár aláírás ellenőrzéséről van szó), de most már a nyilvános kulcsok eredetiségében megbízhatnak, hiszen azokat egy általuk megbízhatónak tartott CA aláírta. Vagyis a kommunikáció nem a nyilvános kulcsok, hanem a tanúsítványok cseréjével kezdődik.

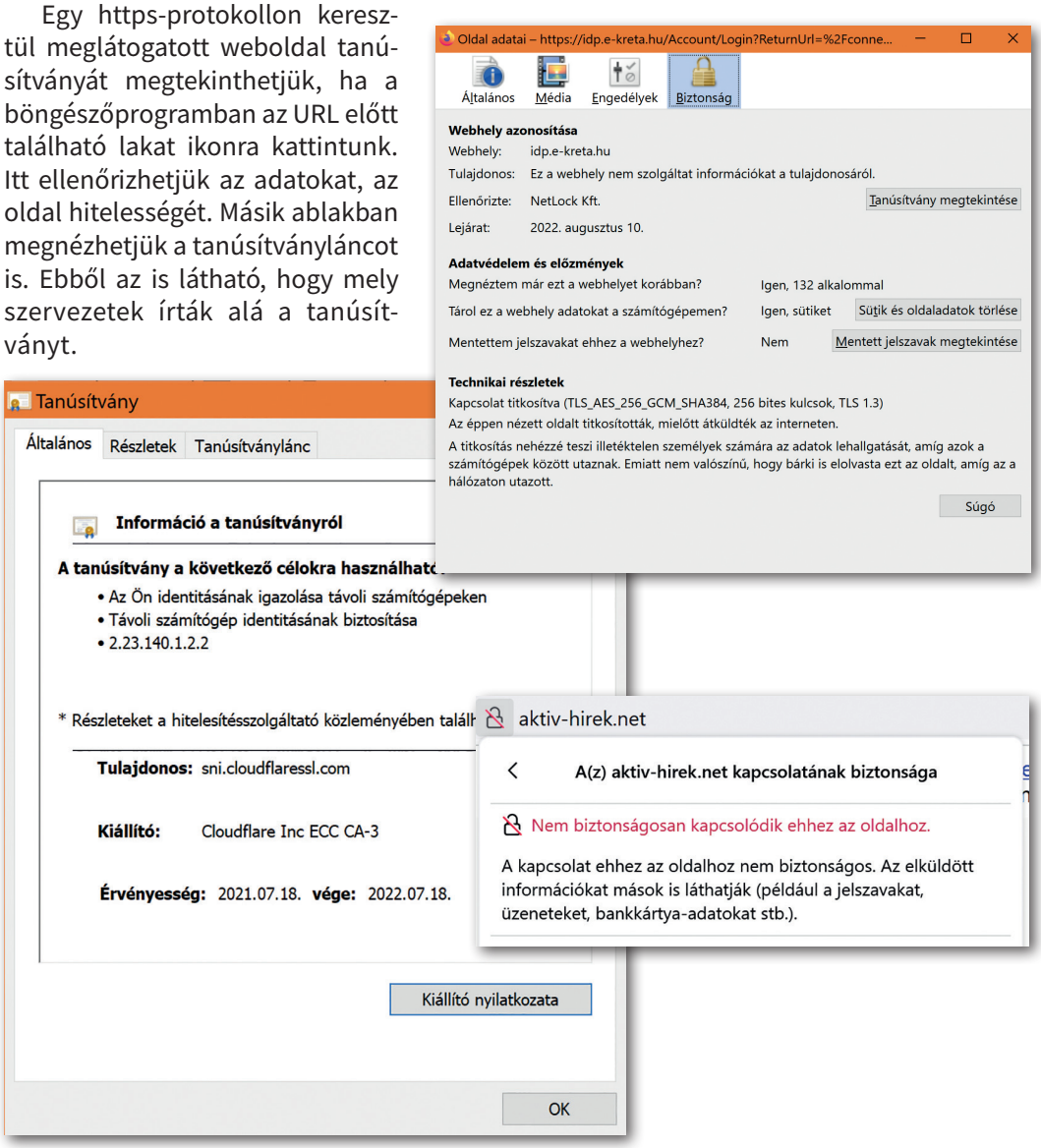
Kérdések, feladatok

1. Milyen előnye van a digitális aláírásnak? Miben különbözik a hagyományos aláírástól? Mit garantál még az aláíró személyén kívül?
2. Nézzünk utána, hogy magánszemélyként, magyar állampolgárként hogyan használhatunk elektronikus aláírást!

Adatvédelem böngészés közben

A https-protokoll mind tartalom-rejtjelezést, mind pedig forrás-/adathitelesítést biztosít, amihez az RSA-tanúsítvány-CA infrastruktúrát használja. A hitelesítést általában „csak féloldalasan” alkalmazzák, ami azt jelenti, hogy csak a szerver hitelesíti magát a kliens felé, de fordítva általában nem. A hitelesítés során a kliens ellenőrzi a szerver tanúsítványát.

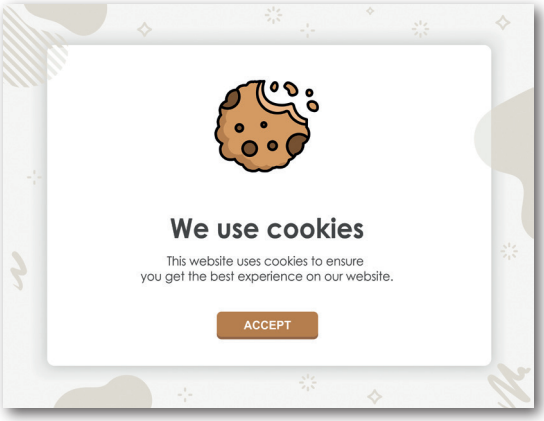
Egy https-protokollon keresztül meglátogatott weboldal tanúsítványát megtekinthetjük, ha a böngészőprogramban az URL előtt található lakat ikonra kattintunk. Itt ellenőrizhetjük az adatokat, az oldal hitelességét. Másik ablakban megnézhetjük a tanúsítványláncot is. Ebből az is látható, hogy mely szervezetek írták alá a tanúsítványt.



Érdemes tehát figyelnünk arra, hogy az általunk meglátogatott weboldal tanúsítványa rendben legyen. Ennek segítségével győződhetünk meg arról, hogy valóban a megfelelő oldallal kommunikálunk.

Ha belépünk egy weboldalra, gyakran tapasztaljuk, hogy el kell fogadnunk a **sütik** (cookies) használatát. Az előző években már megismerkedtünk a süti fogalmával. Tudjuk, hogy ezek kis méretű, adatokat tartalmazó fájlok, amelyeket a weboldal a böngészőprogram segítségével tárol a számítógépünkön. A webszerver és a böngészőprogramok közötti kommunikációt segítik.

A böngészés során többfajta sütit használhatunk.



► Sütik használatára figyelmeztető üzenet

- A **munkamenetsütik** (session cookies) a weboldal működéséhez szükségesek. Nélkülük nem folytatható megfelelő kommunikáció a weboldallal. Érvényességük a weboldalról való kilépéssel lejár, ezért a böngészőprogram bezárásakor automatikusan törlődnek. Ezek a sütik tehát ideiglenesek. Feladatuk sokféle lehet, például a felhasználó azonosítását, a webportál egyes lapjai közötti közlekedést vagy a multimédia-lejátszás beállításait segíthetik.
- Az **állandó sütik** segítségével a weboldallal való kommunikációt tehetjük gördülékenyebbé. Ezeknek köszönhetően oldható meg, hogy ne kelljen beírni a bejelentkezési adatainkat, rögzíteni a kedvelt beállításainkat minden alkalommal, amikor az oldalra bejelentkezünk. Ezek a sütik az oldal elhagyása után is tárolódnak a számítógépünkön, így tudják megőrizni a szükséges adatokat az oldal következő felkereséséig. Egy másik fajtájuk az oldalon való tevékenységeink statisztikai megfigyelését teszi lehetővé. Ez részben az oldal üzemeltetőinek nyújt segítséget a szolgáltatásuk fejlesztéséhez. Lehetséges, hogy céljuk az oldalon megjelenő reklámok egyénre szabása.
- A **harmadik féltől származó** sütiket nem az az oldal hozza létre, amelyiket böngésszük. Ilyen lehet például, amikor a weblapon megjelenő gomb segítségével valamelyik közösségi oldal szolgáltatásait használhatjuk, például lájkolhatjuk, megoszthatjuk a tartalmat. Ilyen az is, amikor egy másik oldal helyezi el a saját hirdetéseit a böngészett oldalon. Használatuk a böngészett oldal számára anyagi hasznot hoz. Ezek a sütik alkalmasak lehetnek a látogató személyének azonosítására.

Az Európai Unió **GDPR**-szabályozásának megfelelően a weboldalaknak tájékoztatást kell nyújtaniuk a sütik használatáról. Amennyiben nem csak a technikailag szükséges sütiket használják, beleegyezésünket kell kérniük ehhez. Ezt a beleegyezést bármikor megváltoztathatjuk. Erre azért van szükség, mert tágabb értelemben véve a böngészési szokásaink, beállításaink, a sütik által tárolt és átadott adataink is tekinthetők személyes adatnak, bizonyos esetekben beazonosítható a személyünk általuk.

► Sütik törlése Mozilla Firefoxban

A böngészőprogramok mindegyikében lehetőségünk van a sütikkel kapcsolatos beállítások kezelésére, a feleslegesek törlésére. Ezt az összes sütire együtt vagy weboldalanként is megtehetjük. Időnként érdemes ezeket az adatokat áttekinteni.

A böngészőprogramok lehetőséget nyújtanak arra is, hogy a nyomkövető weboldalak tevékenységét a beállításokon keresztül korlátozzuk.

Kérdések, feladatok

1. Tekintsük meg egy biztonságos kapcsolaton keresztül felkeresett weboldal tanúsítványát! Nézzük meg a tanúsítványláncot is!
2. Miért lehet gyanús, ha ingyenes szervezet szolgáltatja egy fontos adatainkat bekérő weboldal tanúsítványát?
3. Keressük meg, tekintsük át a böngészőprogramunk adatvédelmi beállításait!
4. Nézzük meg egy kiválasztott weboldalon a cég adatvédelmi nyilatkozatait, beállítási lehetőségeit! Keressük meg, hol lehet megváltoztatni a sütik kezelésére vonatkozó nyilatkozatunkat!
5. Milyen következményekkel járhat, ha a böngészőprogramban valamelyik weboldalhoz tartozó sütiket töröljük?

Kommunikáció az interneten

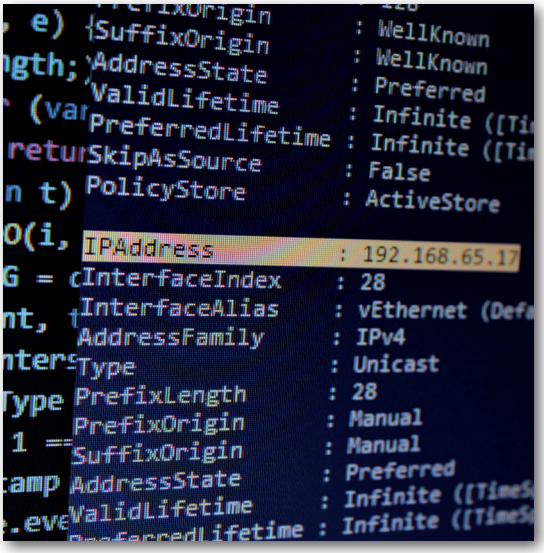
Az előző években megismerkedtünk az online kommunikáció formáival, eszközeivel, fontosabb ismérveivel. Ebben a tanévben ezeket az ismereteket fogjuk pontosítani, elmélyíteni, a folyamatokat kissé alaposabban megvizsgálni.

Számítógépek azonosítása a hálózaton

Az online kommunikáció alapfeltétele, hogy az eszközeink hálózati kapcsolatot tudjanak létesíteni.

A hálózatra kapcsolódó eszközöknek rendelkezniük kell olyan azonosítóval, amely alapján a hálózat szereplői el tudják őket érni, és hozzájuk tudják irányítani az őket illető információkat. Az interneten ezt az egyedi azonosítót **IP-címnek** nevezzük. Az IP-cím egy 4 bájt, azaz 32 bit hosszúságú, kettes számrendszerbeli szám. A könnyebb megjegyezhetőség és kezelhetőség érdekében az IP-cím bájtjait tízes számrendszerbeli számként, egymástól pontokkal elválasztva szoktuk megadni, például: 192.168.65.17.

A technikai fejlődésnek köszönhetően egyre több olyan eszköz jelenik meg, amellyel az internethez tudunk csatlakozni. Ilyen például a mobiltelefon, a különböző okoseszközök. Emiatt a használható IP-címek lassan elfogynak. A fent említett **IPv4-technológia** helyett kidolgozták az **IPv6-szabványt**, ahol a címek nem 32, hanem 128 bitesek. Ennek eredményeként jóval több IP-címet lehet majd kiosztani. Ez a szabvány már ma is elérhető, de bevezetése csak fokozatosan történik, az eddigi címek továbbra is használhatók maradnak.



► IP-cím

A hálózaton az adott eszközt az IP-címe azonosítja, ennek segítségével tud és tudunk vele kommunikálni. Átlagos internethasználóként nehéz lenne azonban minden olyan eszköz IP-címét megjegyeznünk vagy akár csak megadnunk, amellyel valamilyen módon fel szeretnénk venni a kapcsolatot. Azért, hogy erre ne legyen szükség, másképpen is felkereshetünk például egy weboldalt tartalmazó kiszolgálót. Megadhatjuk a doménnevét (domain name: tartománynév). A **doménnév** szavakból, betűkből áll, ezért könnyebben megjegyezhető. Felépítése hierarchikus szerkezetű.

Vegyük a következő példát:

gep.valami.sulihalozat.hu

A doménnév egyes részeit pontok választják el egymástól. Hátulról olvasva ezek a részek egyre kisebb egységeket jelentenek. Jobbról az első rész a *.hu,* az úgynevezett fődomén vagy Top Level Domain. Ez jelenti a legnagyobb szintű egységet, ebben az esetben azt, hogy a domén Magyarországon került bejegyzésre, ehhez a hálózathoz tartozik. Ezen belüli kisebb egység (második szintű domén vagy Second Level Domain) a *sulihalozat,* majd annak kisebb egysége a *valami* hálózat, végül a *gep* az adott számítógép azonosítója a *valami* hálózaton belül. A doménnevek az IP-címekhez vannak hozzárendelve, ezt megfelelő szervereken tartják számon. Ez alapján tudja például a böngészőprogramunk a megfelelő IP-címet és a hozzá tartozó eszközt megtalálni.

Az online kommunikációs szolgáltatások

Személyes kommunikáció

Az internet egyik legrégebb óta használt szolgáltatása az **e-mail.** Szó esett már arról, hogy az igénybevételéhez rendelkeznünk kell e-mail-címmel. Az e-mail-cím felépítéséről tudjuk, hogy két részre tagolódik. Például:

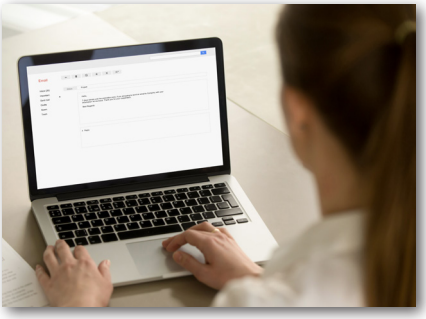
diák@sulihalozat.hu

A két részt a @ karakter választja el, az utána következő rész egy, a postafiókunkat kezelő szervergépet jelöl, az előtte lévő pedig a felhasználónevet, amellyel ezen a gépen regisztráltunk.

Levelezésünket **levelezőszerverek** segítségével bonyolítjuk. Az SMTP- (Simple Mail Transfer Protocol) szerver a kimenő levelek kiszolgálója. A POP3- (Post Office Protocol 3-as verziója) vagy az IMAP- (Internet Message Access Protocol) szerver kezeli a postafiókunkat, tárolja a beérkező leveleinket. A levelek megtekintéséhez valamilyen programra van szükségünk. Ez a webes levelezőrendszerek esetében lehet egy böngészőprogram, más esetben az eszközünkre telepített levelezőprogram.

A **webes levelezést** ingyenes regisztráció után vehetjük igénybe, a szolgáltatásért csak akkor kell fizetni, ha további lehetőségeket, például nagyobb tárhelyet szeretnénk használni. Bárhonnan el tudjuk érni leveleinket, ha van lehetőségünk bejelentkezni a szerverre. Gyakran kapcsolódnak hozzá más szolgáltatások: naptár, állományok tárolására szolgáló tárhely, egyéb, felhőben futtatható programok. Az ingyenes szolgáltatásoknak és tárhelynek sokszor az az ára, hogy a szolgáltató adatokat tárol rólunk. Ezt azért teszi, hogy személyre szabott reklámüzenetekkel kereshessen meg bennünket.

A **telepített levelezőprogramok** a postafiókunkból a gépünkre töltik a leveleinket. Beállítástól függően ezek a szerveren is megmaradhatnak. Ebben az esetben lehetséges a letöltött leveleink offline olvasása, kezelése. Hálózati kapcsolatra csak a levelek küldéséhez és fogadásához van szükség. A levelezőprogramok sok kényelmi funkciót, kiegészítő lehetőséget biztosítanak.



▶ Elektronikus levél írása telepített levelezőprogrammal

Más személyes kapcsolattartási formákkal szemben az e-mail lassabb, kissé körülményesen kezelhető, de a hivatalos ügyintézésnek még mindig ez az egyik legelterjedtebb formája.

A személyes kommunikációra az online tér több olyan formát nyújt, amely az e-maillel ellentétben azonnali reakciók lehetőségét biztosítja. Ilyenek az azonnali üzenetküldő programok és a hang- vagy videókapcsolat kialakítására szolgáló alkalmazások. Milyen alkalmazások tartozhatnak ide?

- **Csevegőprogramok,** melyek segítségével szöveges üzeneteket válthatunk.
- **A VoIP-** (Voice over IP) **alapú telefonálás,** amelynek lényege, hogy a telefonáláshoz nem a hagyományos telefonhálózatok valamelyikét, hanem az IP-alapú (interneten történő) hangátvitel lehetőségét használjuk. Ennek megfelelően a hívás díjazása sem a hagyományos telefonálás díjaival egyezik meg, hanem annál általában olcsóbb.
- A **videókonferenciát** támogató programok.

Ezek a szolgáltatások ma már nem csak külön-külön fordulnak elő. Egyre több olyan komplex program van, amelyben közülük többet használhatunk.

Csoportos kommunikáció

Az online kommunikáció a különböző közösségekkel való kapcsolattartásra is lehetőséget biztosít. Ennek lehetséges formái:

- A **fórum** lehetőséget nyújt egy azonos téma iránt érdeklődő közösség véleménycseréjére.
- A **blog** írója a személyes élményeit, véleményét oszthatja meg az őt követő közösséggel.
- A **levelezőlista** a fórumhoz hasonlóan alkalmas arra, hogy a közösség tagjai egymás között folytassanak megbeszélést, de ebben az esetben ezt elektronikus levelek útján teszik.
- A **közösségi szolgáltatások** felhasználók széles köre számára nyújtanak közös platformot. A szolgáltatáshoz csatlakozók szabadon kommunikálhatnak egymással, tartalmakat tehetnek közzé a többi felhasználó számára. Itt is van lehetőségünk nyílt vagy zárt csoportok kialakítására. Ezek helyettesíthetik a fórumokat és blogokat.
- A **csapatmunka támogatására szolgáló felületek,** a **virtuális osztálytermek** a fenti lehetőségek közül nagyon sokat egyesítenek, alkalmasak a csoportos és személyes kommunikációra mind írásos, mind videós formában, különböző formátumú anyagok megosztására, közös anyagok létrehozására, szerkesztésére, a szervezeten belüli csoportok kialakítására és együttműködésére.

▶ Virtuális osztályterem

Az online kommunikáció folyamatosan átalakul, fejlődik. A lehetőségek egyre bővülnek. A rendszerek egyre szélesebb, változatosabb szolgáltatásokat kínálnak számunkra, akár egy alkalmazáson belül is. Egyre több eszközön, számítógépen, okostelefonon, tableten, okosórán és egyre szélesebb lehetőségekkel használhatjuk ezeket.

Kérdések, feladatok

1. Próbáljuk megtudni egy kiválasztott weboldal IP-címét! Keressünk az interneten megfelelő weboldalt, vagy parancssor indítása után használjuk a *ping* parancsot! Figyeljük meg az IP-cím szerkezetét!
2. Keressük meg a saját számítógépünk IP-címét! Ehhez használhatjuk egy weboldal szolgáltatásait, vagy parancssorból adjuk ki az *ipconfig* parancsot! Milyen egyéb adatokat tudhatunk meg?
3. Nézzünk utána, hogy milyen legfelső szintű domének léteznek! Keressünk olyanokat, amelyek országot, és olyanokat, amelyek más egységet vagy tevékenységi kört jelölnek!
4. Keressük meg a következő kérdésekre a választ a levelezőrendszerünkben!
   a) Milyen műveleteket végezhetünk a beérkezett üzeneteinkkel? Hogyan lehet archiválni, csoportosítani, kiemelni?
   b) Mit jelent a BCC, és hogyan használható?
   c) Hogyan értesülhetünk arról, hogy a levelünk megérkezett a címzett postaládájába?
   d) Milyen lehetőségeink vannak arra, hogy az ismerőseink címét tároljuk? Hogyan kapcsolható ez össze más rendszerekkel?
5. Válasszuk ki, milyen kommunikációs eszközt, konkrét programot használnánk az alábbi helyzetekben! A választást indokoljuk!
   a) Szeretnénk képes beszámolót adni a nagyszüleinknek a nyaralás élményeiről.
   b) Szeretnénk eljuttatni a barátunknak néhány, osztálykiránduláson készült fényképet.
   c) Elektronikus formában szeretnénk kérvényt benyújtani az iskolánk igazgatóságára.

Információk online környezetben

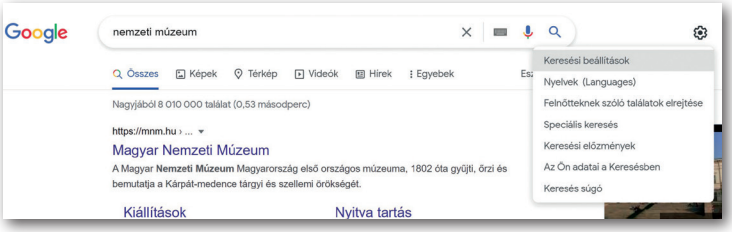
Az online kommunikáció egyik célja a kapcsolattartás, a másik, hogy a számunkra érdekes, szükséges információkhoz hozzájussunk. Az online tér megfelelő eszközöket biztosít ehhez.

Keresés a világhálón

A világháló szolgáltatásaival az előző években már megismerkedtünk. Tudjuk, hogy a weboldalakon, online adatbázisokban rengeteg adat található, és ez a felhalmozott mennyiség napról napra rohamosan növekszik. A weboldalakon, közösségi oldalakon megjelenő információk mindegyikének ellenőrzésére nincs lehetőség, gyakorlatilag bárki bármit közzétehet. Lehetnek kifejezetten félrevezető szándékú tartalmak, amelyek gyakran valamilyen anyagi vagy más jellegű haszonszerzés céljából valótlan, de sokszor figyelemfelkeltő információkat tesznek közzé. Nagyon fontos tisztában lennünk azzal, hogy a világhálón megjelenő információkat mindig kritikusan gondolkodva kell kezelnünk. A világháló segítségével könnyen, gyorsan juthatunk hozzá szinte bármilyen minket érdeklő tartalomhoz, de nem fogadhatunk el minden megjelenő információt fenntartások nélkül. Lényeges, hogy megfelelő stratégiákat ismerjünk, amelyek segítségével viszonylag gyorsan találhatunk rá a megbízható és informatív oldalakra.

Az információ keresésének eszközeivel már megismerkedtünk. Tudjuk, hogy kereshetünk adott témák szerint vagy kulcsszavak alapján, illetve online adatbázisokban. Tanultunk a különböző keresők beállítási lehetőségeiről, a logikai műveletek használatáról.

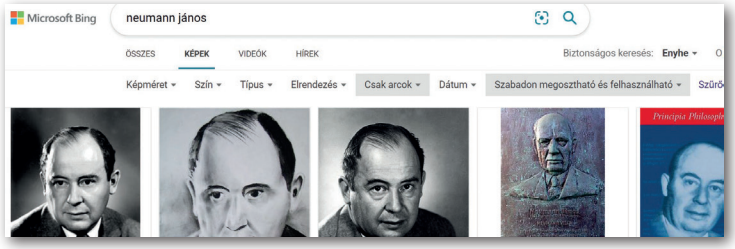
A keresés során további paramétereket állíthatunk be.



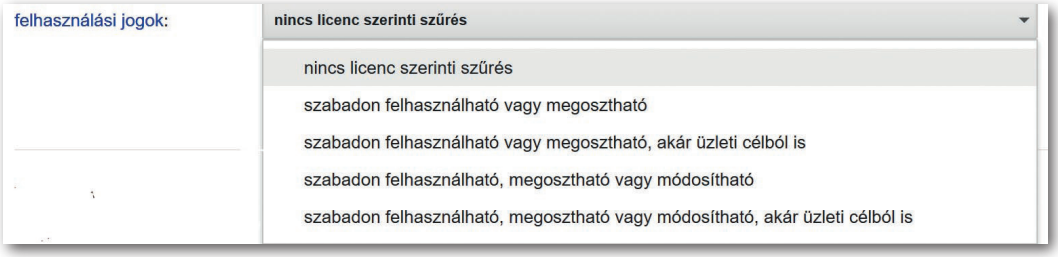
► Keresési beállítások – Google

Megadhatjuk, hogy képet, videót vagy híreket szeretnénk-e keresni. Kereshetünk térképen, beállíthatjuk a nyelvet, szűkíthetjük a találati listát a *speciális keresés* beállításaival.

A világhálón megjelenő tartalmak nem mindegyike **szabadon felhasználható.** A találati listában megjelenő képek nagy részét nem helyezhetjük el saját munkáinkban, saját anyagunkként. A keresők lehetőséget biztosítanak arra, hogy a jogtisztán felhasználható tartalmakra szűkítsük a listát. Érdemes erre figyelni.



► Jogtiszta kép keresése Bing keresővel



▶ Felhasználási jogok beállítása – Google

Léteznek olyan online képgyűjtemények, amelyekben nagy mennyiségű, ingyenesen felhasználható, jó minőségű kép közül válogathatunk. Ilyen képeket találhatunk például a *pixabay.com/hu/* vagy az *unsplash.com* oldalakon.

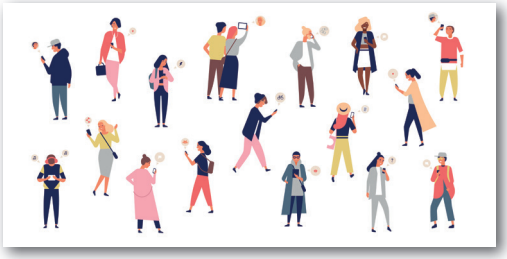
A jogtiszta felhasználás mellett érdemes arra is figyelnünk, hogy a megtalált tartalmak mennyire tekinthetők hitelesnek. Milyen kérdéseket érdemes ezzel kapcsolatban mérlegelnünk?

- Milyen társaság vagy személy működteti az adott weboldalt? Tartozik-e valamilyen megbízható szervezethez?
- Keressük meg az oldal szerzőjét, szerzőit! Nézzünk utána, mennyire megbízhatók, van-e más anyaguk hasonló témában!
- Mikor keletkezett az oldal? Mikor frissítették utoljára?
- Vannak-e az oldalon hivatkozások, amelyek a témának megfelelő tartalmakra mutatnak?
- Vannak-e forrásmegjelölések?
- Milyen az oldal tartalmi és formai minősége? Nyelvtan és helyesírás szempontjából megfelelő-e?

Egyes keresők rendelkeznek azzal a lehetőséggel is, hogy csak tudományosan megbízható tartalmak találatait jelenítsék meg. Ilyen például a Google tudományos keresője, a *Google Tudós (scholar.google.com)* vagy a BASE (Bielefeld Academic Search Engine, www.base-search.net).

Feladatok

1. Keressük meg, melyik volt 2021-ben az év madara! Töltsünk le róla olyan .jpg formátumú képet, amely szabadon felhasználható, és az elmúlt évben készült!
2. Alakítsunk 3-4 fős csoportokat, és válasszunk egy európai nagyvárost! Készítsünk 6–8 oldalas prezentációt, amely bemutatja a várost! Térjünk ki a népességére, területére, nevezetességeire és arra, hogy hogyan lehet oda eljutni lakóhelyünkről! Minden felhasznált adatnál, képnél jelöljük meg azt az oldalt, ahonnan származik!
3. Keressünk adatokat arról, hogy hogyan változott a Föld éghajlata az elmúlt tíz évben! Igyekezzünk minél hitelesebb információforrásokat találni!



Előzetes tanulmányainkban már megismerkedtünk a mobil informatikai eszközök jellemzőivel, az okostelefonok biztonságos használatával, a mobiltanulást támogató alkalmazásokkal, sőt saját mobilalkalmazást is készítettünk.

Ebben a fejezetben egy olyan alkalmazással foglalkozunk, amely az együttműködést, a projektmunkát és a közös feladatmegoldást segíti. Ezt nemcsak iskolai projektek megvalósítása során, hanem az élet számos más területén felhasználhatjuk, legyen az munka vagy akár egy rendezvény szervezésével járó feladat.

Projektmunka mobil informatikai eszközökkel

Projekt tervezése és megvalósítása

Tudjuk, hogy egy **projekt** megvalósítása során egymással együttműködve, feladatokat kitűzve, azokat egymással megosztva dolgozunk egy adott cél elérése érdekében.

A projekt elindulásakor meg kell terveznünk, hogy milyen **tevékenységeket** és milyen **sorrendben** fogunk elvégezni.

A projekt **mérföldköveit** is érdemes meghatározni. Ezek olyan állomások, amelyek a projekt megvalósítása során rendkívül fontos szerepet töltenek be, és jól szemléltetik a projekt előrehaladását. Ilyen lehet az, ha elkészül egy termék. A mérföldkövek és az azokhoz tartozó feladatok **határidejét** úgy kell meghatároznunk, hogy a teljes projektet az elvárt végső határidőre befejezhessük.

A feladatokhoz **felelősöket** kell hozzárendelnünk, illetve azt is meg kell határoznunk, hogy a feladat megvalósításához szükséges-e valamilyen **tárgyi feltétel** (például speciális eszköz) vagy esetleg **anyagi forrás** (például osztálykirándulás tervezésénél a programok költségei).

Fontos látnunk, hogy az egyes tagok milyen **feladatokon** dolgoznak, illetve azok milyen **állapotban** (státuszban) vannak (például nyitott, folyamatban lévő vagy befejezett).

A feladatok között lehetnek **egymásra épülők,** amikor bizonyos feladatok megoldásába csak akkor tudunk belekezdeni, ha egy **előfeltétel** (egy másik feladat befejezése) teljesült. Előnyös tehát, ha a feladatok közti kapcsolatot beállíthatjuk, illetve azt valamiképpen megjeleníthetjük.

A megoldandó feladatok **nehézsége,** komplexitása változó. Ezért fontos lehet, hogy a feladatokhoz nehézségi szintet vagy pontszámot rendeljünk. A résztvevők teljesítménye így mérhető lesz azáltal, hogy mennyi és milyen nehézségű feladat megoldásából vették ki a részüket.

Jó, ha a feladatokhoz **megjegyzéseket** vagy akár **csatolmányokat** tudunk társítani. A **címkézés** is hasznos funkció. A címkékkel csoportosíthatjuk, kategorizálhatjuk a feladatokat, illetve azok alapján könnyen kereshetünk, szűrhetünk a feladatok listájában. Sőt, akár a címkékkel jelölhetjük a feladatok nehézségi szintjét. Ha egy feladat megvalósításánál valamilyen előre nem látott probléma merült fel, szintén jelezhetjük azt a megfelelő címkékkel.

Lehet, hogy a projekt előrehaladását meg szeretnénk osztani valakivel úgy, hogy a feladatokat ne módosíthassa, csak megfigyelő szerepe legyen. Ezért szükség lehet különböző **jogosultsági szintekre** is. Erre példa, hogy csak a projekt vezetője határozhatja meg azt, hogy kik vesznek részt a projektben, a tagoknak nincs lehetőségük újabb személyek bevonására.

Amikor elértük az adott célt, akkor be kell mutatnunk a **projekt eredményeit,** majd érdemes **értékelni** a projekt sikerességét. Ennek során levonhatjuk a tanulságokat arra vonatkozóan, hogy esetleg min lehetne változtatni egy következő projekt megvalósítása során.

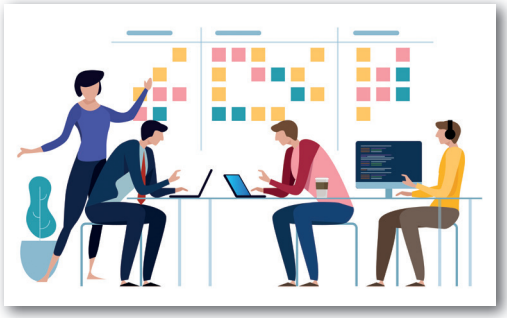
Projektmenedzsment alkalmazások

A projektmunka adminisztrálását célszerű egy speciális számítógépes alkalmazás segítségével végezni, amelyet **projektmenedzsment szoftvernek** nevezünk.

Az alkalmazás segítségével minden résztvevő valós időben láthatja, hogy ki mivel foglalkozik, milyen feladatokba kezdett bele, illetve mit fejezett már be.

Emellett a határidőkről figyelmeztető üzeneteket kaphatunk, ami segíti a projekt előrehaladását. A feladatok státuszát is egyszerűen módosíthatjuk.

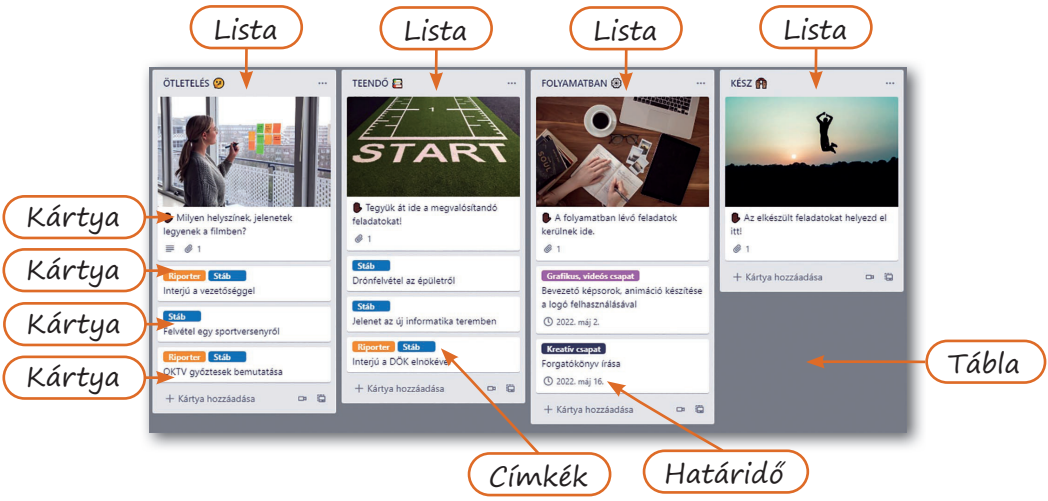
A projektmunkát támogató számítógépes alkalmazások más-más felületen, felépítésben teszik lehetővé a feladatok adminisztrálását. Abban is különböznek, hogy az imént megismert funkciók közül melyeket támogatják, és melyeket nem.



Ügyeljünk arra, hogy a legtöbb esetben az alkalmazások ingyenesen csak korlátozottan használhatók, a teljes funkcionalitás eléréséhez előfizetésre vagy prémiumtagságra lehet szükség.

A következőkben egy magyar nyelven elérhető, egyszerűen használható és hatékony alkalmazással ismerkedünk meg.

A **Trello** környezetben (https://trello.com/) munkaterek, táblák, listák és kártyák segítségével adminisztrálhatjuk a projektünket, akár magyar nyelven is.



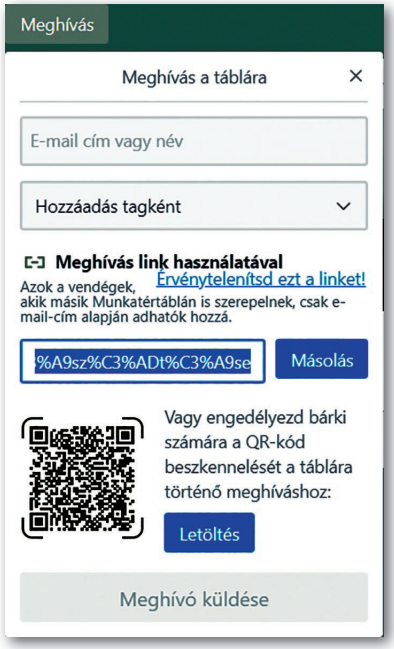
▶ Mintaprojekt (iskolát bemutató film készítése) a Trello alkalmazásban

A regisztráció után nevet kell adnunk a munkaterünknek. A **munkatér** a projektben részt vevők fő navigációs központja.

A munkatéren belül lehetőség van **táblák** létrehozására. A táblák szolgálnak az egyes feladatok, teendők rendszerezésére. Egy tábla akár egy teljes projekt feladatait is tartalmazhatja. Bonyolultabb projektek esetén érdemes több táblát használni egy munkatéren belül.

A táblát azok tudják szerkeszteni, akiket a *Meghívás* gombra kattintva meghívtunk. Lehetőségünk van arra is, hogy egy hivatkozás segítségével hívjuk meg a tagokat. A link vagy a QR-kód megosztása után a projektben részt vevők önállóan csatlakozhatnak a táblákhoz.

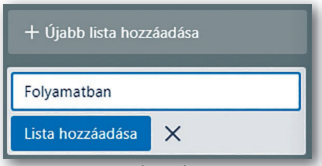
A táblákon elhelyezhetünk **listákat** is. A listák segítségével tudjuk a feladatokat csoportosítani. A lista címét úgy adjuk meg, hogy az megfelelően utaljon arra, milyen jellegű feladatokat fog tartalmazni. Például *ötletelés, teendő, folyamatban, kész.*



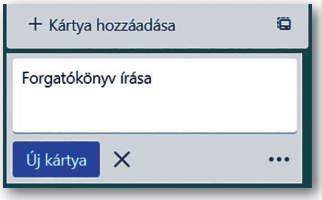
▶ Tagok meghívása

A listákon belül elhelyezett **kártyák** a tábla legkisebb egységei. Ezek tartalmazzák a tényleges tennivalókat, feladatokat vagy akár a megvalósítási ötleteket. Létrehozásukhoz a lista alján lévő *+ Kártya hozzáadása* hivatkozásra kell kattintanunk.

A **kártyákat** fogd és vidd módszerrel **mozgathatjuk** az egyes listák között. Ha például egy feladatot elkezdünk megoldani, akkor azt áthúzhatjuk a *Teendő* című listából a *Folya-*



▶ Lista hozzáadása



▶ Kártya hozzáadása

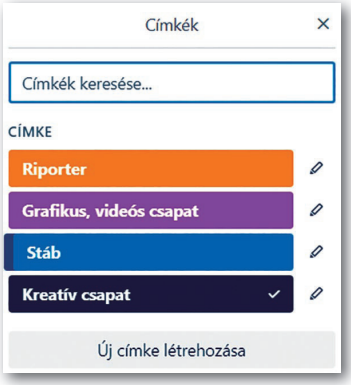
▶ Kártya beállítása

matban című listába. Ha pedig végeztünk a feladattal, jelezhetjük úgy, hogy áttesszük azt a *Kész* című listába. Ha minden csapattag így jár el, akkor mindig pontosan láthatjuk, hogy az egyes feladatok melyik állapotban (státuszban) vannak.

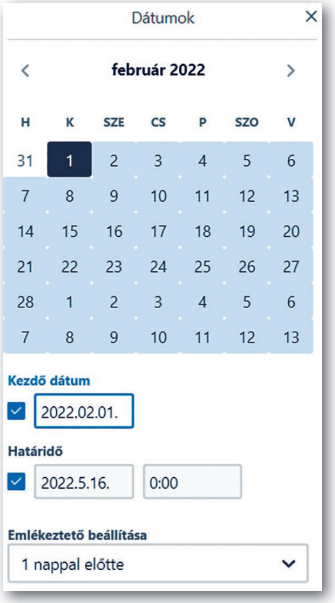
A kártyára kattintva további fontos beállításokat végezhetünk.

A *Csatlakozz* gombbal hozzárendelhetjük magunkat a kártyához, így felkerülünk a tagok közé. A tagokat a *Tagok* pontra kattintva is beállíthatjuk, így akár mást is hozzárendelhetünk az adott kártyához. A kártyához felvett tagok képei vagy monogramjai a kártya jobb alsó sarkában láthatóak lesznek.

A *Címke* hivatkozásra kattintva beállíthatjuk a használni kívánt címkéket. Mindegyik címkéhez választhatunk egy színt, illetve megadhatjuk a címke szövegét is.



▶ Címkék létrehozása



▶ Dátumok beállítása

▶ Kitöltés beállítása

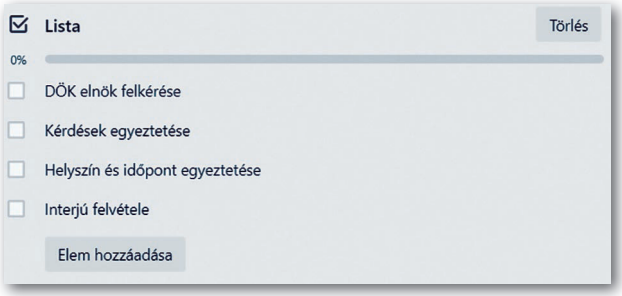
A *Dátumok* megadásánál mind a kezdő dátumot, mind a határidőt kijelölhetjük, illetve beállíthatjuk, hogy a rendszer mikor küldjön emlékeztetőt a határidőről (például határidő előtt egy nappal).

A *Csatolmány* segítségével számítógépen/mobileszközön lévő állományokat csatolhatunk a kártyához, vagy egy már megosztott dokumentum linkjét is elhelyezhetjük rajta.

A *Kitöltés* opcióval a kártyát átszínezhetjük, vagy borítóképet tölthetünk fel rá.

Amennyiben a projekt megvalósítása több helyszínen zajlik, a *Hely* menüpont segítségével érdemes beállítani a helyszínt. Ekkor egy Google-térkép jelenik meg a kártya adatlapján, jelölve a pontos helyszínt.

Ha az adott feladat jól elkülöníthető alfeladatokból áll, érdemes belőlük listát létrehozni a *Lista* hivatkozás követésével.



► Kártya beállítása

Amikor végeztünk egy-egy feladattal, elég lesz rákattintanunk a sor elején lévő jelölőnégyzetre.

Láthatjuk tehát, hogy a *Trello* alkalmazás sok szinten támogatja a projektmunka, illetve csoportmunka szervezését és megvalósítását. Nincs más hátra, mint hogy kipróbáljuk a gyakorlatban is!

Feladatok

1. Válasszunk ki közösen egy szimpatikus projektmenedzsment szoftvert, amely mobilapplikációval rendelkezik! Használhatjuk a *Trello* alkalmazást, de megvizsgálhatunk más, hasonló tudással rendelkező alternatívákat is (például *Asana, ClickUp*). Feltétlenül nézzük meg, hogy az adott alkalmazás (ingyenesen) hány fős projektek kezelésére alkalmas, és olyat válasszunk, amely megfelel az elvárásoknak. Ha például az egész osztály dolgozik a projekten, akkor nem lesz jó egy tizenöt felhasználóig használható alkalmazás.
2. Próbáljuk ki először a rendszer lehetőségeit egy egyszerűbb feladat kapcsán! Tartsuk nyilván például azt, hogy egy tetszőlegesen kiválasztott tantárgy aktuális beadandójával kapcsolatban ki melyik munkafázisban tart (például *még nem kezdte el, anyaggyűjtés, megvalósítás folyamatban, kész*)!
3. Vitassuk meg közösen, hogy hogyan lenne érdemes az adott környezetben lebonyolítani a projektet! A téma lehet például a következő osztálykirándulás megszervezése. Néhány ötlet a megvalósításhoz:
   - Legyen lehetőség egyénileg javaslatot tenni az osztálykirándulás helyszínére. A javaslathoz lehessen dokumentumot csatolni, amely röviden bemutatja a helyszín jellegzetességeit, és tartalmaz néhány programötletet.

- Az ötletek alapján szavazással válasszuk ki a három legígéretesebb helyszínt!
- A helyszínekhez rendeljünk csoportokat! A csoportok feladata, hogy az általuk választott vagy kisorsolt helyszínre megtervezzék az osztálykirándulást. Állapodjunk meg előre abban, hogy mekkora költségkerettel lehet számolni! Legyen egy alacsonyabb és egy magasabb költségkeret is megadva.

- Minden csoportban legyen egy olyan csapat, amelynek tagjai a programlehetőségeket gyűjtik össze és dokumentálják. A következő csapatnak megfelelő szállás- és étkezési lehetőségeket kell találnia. A harmadik csapatnak a közlekedés módját kell megterveznie, alternatív lehetőségek (például buszbérlés, utazás vonattal, tömegközlekedés használata) figyelembevételével.
- Minden csoport készítse el a kirándulástervét az előállt dokumentumok alapján két változatban, a két költségkeret figyelembevételével!
- A csoportok mutassák be a terveket, majd döntsük el együtt, hogy melyik terv legyen megvalósítva!

Algoritmizálás és programozási nyelv használata

Az *Algoritmizálás és programozási nyelv használata* című fejezet átfogó ismétléssel kezdődik. Ebben a kötetben már megismerkedünk az elemi típusalgoritmusok strukturált programozásban használatos formájával. A komolyabb, nehezebb programozási ismeretek közül elsőként a fájlkezelés kerül sorra, mert remekül használható az összetett típusalgoritmusok megismerésekor. Az algoritmusok közül erősebb hangsúlyt kapnak a középiskolai szintű feladatokban gyakrabban előfordulók.

Az emelt szintű ismereteket taglaló témaköröket követően a fejezet tárgyalási módja változik. Az objektumorientált programozásról és a grafikus felület programozásáról szólva a sok rövid példa helyett egy-két feladat részletesebben ismertetett megoldására hagyatkozunk, melyekben ismerkedni kezdünk az olyan problémamegoldási attitűddel és módszerkészlettel, amilyet a nagyobb lélegzetű feladatok megoldása gyakran megkíván.

Vittük valamire – Szekvenciák, elágazások és a feltételes ciklus

Kilencedik és tizedik évfolyamon jócskán belekóstoltunk a programozásba, és már egészen klassz dolgokra tudjuk rávenni a számítógépet – ebben és a soron következő három leckében először ezeket az ismereteket ismételjük át.

A bennünket körülvevő rengeteg számítógép közül a digitáliskultúra-órákon minket épp azok érdekelnek leginkább, amelyekről látszik, hogy számítógépek: a laptopok és az asztali számítógépek. Mindazonáltal tudjuk, hogy éppúgy programok működtetik a háztartási gépeinkben, járműveinkben, egyéb eszközeinkben lévő számítógépeket is, és az sem titok, hogy mára csak a legegyszerűbb gépeinkben nincs számítógép.

Az említett programokat sokféle nyelven írhatjuk – itt a könyvben történetesen a Pythonban, ha úgy tetszik, pythonul fogalmazzuk meg a programjainkat. Legyen azonban szó bármelyik programozási nyelvről, ha elég messziről nézzük őket, igencsak hasonlóak. Az előző mondatnak csak látszólag mond ellent az a megjegyzésünk, hogy az egyes nyelvek bizonyos feladatok elvégzésére alkalmasabbak lehetnek a többinél. Az újabb és újabb programozási nyelvek, illetve környezetek sokszor a visszatérő problémák kész megoldását nyújtják függvények, eljárások formájában.

A programok mindig utasításokból állnak. A legtöbb nyelv használatakor – ha másképp nem kérjük – a számítógép ezeket az utasításokat szépen sorra veszi, és egymás után – programozóul úgy mondjuk, hogy **szekvenciában** – végrehajtja őket.

A sorszámokat nem írjuk be, csak a könyvben számozzuk, mert így könnyebb elmondani, hogy melyikről beszélünk.

```
1. #!/usr/bin/env python3
2.
3. print('Idén is lesz programozás.')
4. print('Hát nem remek? ;)')
```

Ez a sor csak Linuxon és macOS-en kell, Windowson nem fontos. Mostantól a könyvben nem írjuk ki.

Már a legegyszerűbb programok is képesek lehetnek a felhasználóval való kommunikációra. A felhasználó válaszait (is) változókban tároljuk. A **változók** értéke a változó nevének említésével kiolvasható.

Gyakori, hogy egy változó értékétől függően mást és mást csinál a programunk. Ilyenkor **elágazás** van a programban, aminek csak az egyik ágát szeretnénk végrehajtani.

Változót hozunk létre, és elhelyezzük benne a felhasználó válaszát.

```
1. print('Idén is lesz programozás.')
2. vélemény = input('Örülsz? (i/n) ')
3. if vélemény == 'i':
4.     print('Hát én is!')
5.     print('Jaj, de jó!')
6. elif vélemény == 'n':
7.     print('Hüpp.')
8. else:
9.     print('Nem értem. Pedig igazán próbálkoztam.')
10. print('Pápá.')
```

Ez a két sor akkor fut le, ha a felhasználó „i”-t válaszolt.

Ez akkor, ha „n”-t.

Ez minden egyéb esetben.

Ez pedig mindig lefut, mert nincs behúzva, azaz az elágazás után következik.

1. példa: Cica vagy kutya?

Írjuk meg azt a programot `cicakutya` néven, amelyik megkérdi a felhasználót, hogy a program cica vagy kutya legyen-e! Ha a felhasználó cicát szeretne, akkor a program kérjen tejet, és írja ki, hogy „Nyaú!”. Kutya esetén persze csontra lesz szükség, a megnyilvánulás pedig „Vaú!”.

```
1. állat = input('Cica vagy kutya legyek? (c/k) ')
2. if állat == 'c':
3.     print('Tejet, ha lehet.')
4.     print('Nyaú!', end='')
5. elif állat == 'k':
6.     print('Egyet mondok, adjál csontot!')
7.     print('Vaú!', end='')
```

Ha valamilyen ismétlődő feladatot szeretnénk végeztetni a számítógéppel, **ciklust** szervezünk. Az első ciklusunk az **elöltesztelő feltételes ciklus** vagy `while`-ciklus. Ebbe a ciklusba akkor lépünk be, ha a belépés feltétele igaz, és addig maradunk benne, amíg ez a feltétel teljesül.

Ha a felhasználó „i”-t válaszolt, belépünk a ciklusba, és cincogunk.

```
1. legyen_cincogás = input('Üdv, én egy egér vagyok. Cincogjak neked? (i/n) ')
2. while legyen_cincogás == 'i':
3.     print('Cin-cin!')
4.     legyen_cincogás = input('Még? (i/n) ')
5. print('Szia.')
```

Ha itt „n”-t (pontosabban: nem „i”-t) válaszolunk, akkor a következő ismétlődésnél a 2. sorban nem teljesül a feltétel, és nem maradunk a ciklusban.

2. példa: A harmincéves háború időtartama

Írjunk programot, amely addig kérdezgeti, hogy hány évig tartott a harmincéves háború, amíg a felhasználó ki nem találja! Ha a programozási nyelvünk miatt szükség van rá, ne felejtsük megfelelő típusúra alakítani a választ! Segítségül itt az algoritmus mondatszerű leírása:

```
program haboru30:
    válasz = speciális érték, ami biztosan nem 30
    ciklus amíg válasz <> 30
        be: válasz
    ciklus vége
    ki: dicséret
program vége
```

Amikor már működik a programunk, itt az ideje továbbfejleszteni. Ha a felhasználónk tippje hibás, írjuk ki, hogy kisebb vagy nagyobb-e a megfelelő érték! Ha pedig harmadjára sem találta el, megsúghatjuk neki, hogy mettől meddig tartott a szóban forgó háború.

```
1. válasz = None
2. próbálkozások = 0
3.
4. while válasz != 30:
5.     if próbálkozások >= 3:
6.         print('Súgok: 1618-tól 1648-ig tartott.')
7.     válasz = input('Hány évig tartott a harmincéves háború? ')
8.     válasz = int(válasz)
9.     próbálkozások += 1
10.    if válasz > 30:
11.        print('Rövidebb volt.')
12.    elif válasz < 30:
13.        print('Hosszabb volt.')
14.
15. print('Ügyes! Nem gondoltam volna...')
```

Létrehozzuk a változót (mert a negyedik sorban már kelleni fog), de nem adunk neki értéket.

Vittük valamire – Bejárható objektumok és a bejárós ciklus, eljárások és függvények

Az előző leckében a feltételes ciklussal is foglalkoztunk. A másik ciklusunk – merthogy Pythonban kétféle van – a **bejárós** vagy `for`-ciklus. A feltételes ciklusnak adnunk kell egy bejárható objektumot, amelynek az elemein végiglépdelhet, azaz amit bejárhat. Eddig négyféle bejárható objektummal ismerkedtünk meg. Lássuk őket sorban:

A **range típusú objektumok** egy számsort tartalmaznak, és a `range()` függvénnyel tudunk ilyet előállítani. A bejárós ciklus **ciklusváltozója** range-objektum bejárásakor az egyes számokat kapja értékül.

3. példa: Szökőévek I. Ferenc uralkodásától napjainkig

Írjunk programot, amely megadja I. Ferenc József trónra lépésétől az idei évig tartó időszak szökőéveit!

```
1. idei_év = 2022
2. for év in range(1848, idei_év+1):
3.     if év % 4 == 0 and \
4.         (év % 100 != 0 or év % 400 == 0):
5.         print(év, '.', sep='')
```

Listák és karakterláncok

A **listák** összetartozó adatokat tartalmaznak. Amikor listát járunk be `for`-ciklussal, akkor a ciklusváltozó az egyes listaelemeket kapja értékül.

A **karakterláncok,** azaz stringek a listákhoz hasonlóan viselkednek. Egy string bejárásakor a ciklusváltozó az egymás utáni karakterek értékét veszi fel. Karakterláncok és listák között fontos különbség, hogy csak a listák módosíthatók. Új listaelemek hozzáadásával bővíthetők, a meglévő elemeik cserélhetők vagy törölhetők. Mindez a stringek esetében nem lehetséges.

Az alábbi kód a lista és a karakterlánc adattípus pár tulajdonságát mutatja be.

```
1. szöveg = input('Írj ide egy szót kisbetűkkel! ')
2. szöveg = szöveg.upper()
3.
4. lista = []
5.
6. for betű in szöveg:
7.     print('A(z)', betű, 'kerül a listába.', end=' ')
8.     lista.append(betű)
9.     print('A lista értéke most: ', lista)
```

A stringeknek van néhány remek tagfüggvényük. A „python string methods" kifejezésre keresve találunk róluk információt.

A listáknak is van néhány remek tagfüggvényük. Hasonló kereséssel ezeket is megleljük az interneten.

Feladat

Keressük meg az interneten, hogy mi minden közös a listák és a karakterláncok kezelésében, és nevezzünk meg néhány eltérést is!

4. példa: A három kismalac háza

Listákból használtunk kétdimenziósakat is – ezek olyasmik, mint a táblázatok. Lássunk egy olyat (`malacok.py`), amely a három kismalac nevét és házának anyagát tárolja.

```
1. malacok = [['Töfi', 'szalma'],
2.            ['Röfi', 'fa'],
3.            ['Döfi', 'kő']]
```

A „nagy” lista három kisebb listát tartalmaz.

Amikor egy ilyen listát bejárunk, a ciklusváltozóba az épp aktuális kis lista kerül.
Írjuk ki egy-egy sorba az egyes malacok nevét és a házuk anyagát! Amelyik malacnak „kő”-ből készül a háza, amellé írjuk oda azt is, hogy a farkas nem tudja bántani! Teszteljük a programot úgy, hogy a listába újabb kőházas malacokat veszünk fel!

```
4. for malac in malacok:
5.     print(malac[0], ' házának anyaga: ' , malac[1], '.', sep='', end='')
6.     if malac[1] == 'kő':
7.         print(' A farkas nem tudja bántani ', malac[0], 't.', sep='')
8.     else:
9.         print()
```

Bár a programunk remekül működik, egy idő után nehéz lehet követni, hogy mit is jelent a `malac[0]` és a hasonló jelölések. A **szótárak** pont ezen a problémán segítenek. A szótárakban az egyes objektumoknak különféle tulajdonságait tárolhatjuk:

```
1. malacok = [{'név': 'Töfi', 'ház_anyag': 'szalma'},
2.            {'név': 'Röfi', 'ház_anyag': 'fa'},
3.            {'név': 'Döfi', 'ház_anyag': 'kő'}]
```

Egy-egy malacot egy-egy szótárban tárolunk.

A három szótár az előző „nagy” listában van.

Módosítsuk úgy a kétdimenziós listát tartalmazó malacos programot, hogy ezúttal szótárakkal dolgozzon!

```
4. for malac in malacok:
5.     print(malac['név'], ' házának anyaga: ' , malac['ház_anyag'], '.', sep='', end='')
6.     if malac['ház_anyag'] == 'kő':
7.         print(' A farkas nem tudja bántani ', malac['név'], 't.', sep='')
8.     else:
9.         print()
```

A szótárak szintén bejárható objektumok. Bejárhatók úgy, hogy a kulcsaikat kapjuk meg (az előző példában „név” és „ház_anyaga”), de bejárhatók érték szerint vagy épp mindkettőt elkérve. Lássunk egy példát:

A szótárakban a kulcsok egyediek – nem lehet két ,,név'' vagy két ,,magasság'' egyetlen szótárban.

```
1. ember = {'név': 'Debóra', 'magasság': 167, 'IQ': 131}
2.
3. for kulcs in ember.keys(): # vagy csak: in ember
4.     print(kulcs, 'értéke:', ember[kulcs])
5. print()
6.
7. for érték in ember.values():
8.     print(érték)
9. print()
10.
11. for kulcs, érték in ember.items():
12.     print(kulcs, 'értéke:', érték)
```

A kulcs szerinti bejáráskor a ciklusváltozóba az épp aktuális elem kulcsa kerül. A kulcs ismeretében megkaphatjuk az értéket is.

Így a kulcsot és az értéket is megkapjuk.

5. példa: Kedvenceink

Írjunk egy programot `kedvencek` néven, amely egy-egy szótárban tárolja három kedvenc vloggerünket, előadónkat, tanárunkat vagy sakknagymesterünket! Egy szótárnak három kulcsa legyen:

- a tárolt ember neve,
- legjelentősebb teljesítménye a szakmájában és
- az említett teljesítmény évszáma.

A három szótárat helyezzük el egy listában, majd járjuk be a listát, és írjuk ki, amit az egyes emberekről tudunk!

```
1. nagymesterek = [{'név': 'Polgár Judit',
2.                  'teljesítmény': 'Garri Kaszparov legyőzése',
3.                  'év': 2002},
4.                 {'név': 'Kempelen törökje',
5.                  'teljesítmény': 'Napóleon legyőzése (sakkban)',
6.                  'év': 1809},
7.                 {'név': 'Beth Harmon',
8.                  'teljesítmény': 'TV-sorozattá lett az "élete"',
9.                  'év': 2020}]
10.
11. for nagymester in nagymesterek:
12.     print(nagymester['név'], ': ',
13.           nagymester['teljesítmény'], ' (',
14.           nagymester['év'], ')', sep='')
```

A három szótár

Listában vannak.

Így nem kezd új sort a print().

Ha a programunk szépen működik, akkor úgy bővítjük, hogy a felhasználónak legyen lehetősége új embereket felvenni a listába. A listát az új emberek felvétele előtt és után is ki akarjuk írni. Ha egy programban több helyen csinálnánk ugyanazt, akkor eljárást vagy függvényt készítenénk a feladat elvégzésére. A kettő között az a különbség, hogy a függvénynek van visszatérési értéke, az eljárásnak nincs. Sok nyelv, így a Python sem különbözteti meg az eljárást a függvénytől: az eljárást egyszerűen olyan függvénynek tekinti, amelynek nincs visszatérési értéke.

Alapvetően nem jó, ha az eljárások és a függvények a főprogram változóihoz nyúlnak. Okosan tesszük, ha paraméterként átadjuk számukra azokat a változókat, amikkel dolgunk van.

Alakítsuk át a főprogram 11–14. sorát! Most, hogy eljárás lett belőlük, érdemes lehet a program elejére helyezni őket.

```
1. def nagymestereket_listáz(nmk):
2.     for nm in nmk:
3.         print(nm['név'], ': ', nm['teljesítmény'], ' (', nm['év'], ')', sep='')
```

Ezzel megszületik a `nagymestereket_listáz` eljárásunk, amelynek a paramétere az „`nmk`” nevű lista. Az eljárás bejárja a listát, és kiírja az elemeit. Elterjedt szokás – bár a Pythonban nem kötelező – az eljárásokat és a függvényeket a program elején elhelyezni. Az eljárások lefuttatása, azaz hívása az eljárás nevével történik. A programunkba szúrjuk be az alábbi sort!

```
11. nagymestereket_listáz(nagymesterek):
```

Híváskor a „nagymesterek” listát adjuk át – de az eljárás már „nmk” néven fog a megkapott listával dolgozni.

Ha most is minden szépen működik, akkor átgondoljuk, hogy egy új ember megadásához mit kell tenni:

- három adatot be kell kérni,
- az évszámot számmá kell alakítani,
- az adatokból egy szótárat készíteni és
- a szótárat a lista végére fűzni.

Az első három feladat megoldását egy függvénybe szervezzük. A függvényt csak egyszer fogjuk lefuttatni, de olvashatóbb lesz így a kód, jobban elkülönül, hogy a program melyik része mit csinál – ez a másik eset, amikor eljárást, függvényt használunk. A függvényünknek nincs paramétere, de van visszatérési értéke – hiszen ettől függvény –, mégpedig az új ember adatait tartalmazó szótár.

```
1. def adatokat_bekér():
2.     név = input('Mi a neve? ')
3.     teljesítmény = input('Mi a legjelentősebb teljesítménye? ')
4.     év = int(input('Melyik évben érte ezt el? '))
5.     szótár = {'név': név, 'teljesítmény': teljesítmény, 'év': év}
6.     return szótár
```

A függvényt a programunk végén hívjuk. A főprogram (amelyet megelőz a függvény és az eljárás definíciója, valamint a lista eredeti változata) most ilyen:


```
22. nagymestereket_listáz(nagymesterek)
23.
24. új_ember = adatokat_bekér()
25. nagymesterek.append(új_ember)
26.
27. print('A nagymesterek listája ilyen lett:')
28. nagymestereket_listáz(nagymesterek)
```

Már csak annyi dolgunk maradt, hogy egy ciklussal újra meg újra megkérdezzük, hogy akar-e még a felhasználónk újabb embert megadni. Íme a teljes kód:

```
 1. def adatokat_bekér():
 2.     név = input('Mi a neve? ')
 3.     teljesítmény = input('Mi a legjelentősebb teljesítménye? ')
 4.     év = int(input('Melyik évben érte ezt el? '))
 5.     szótár = {'név': név, 'teljesítmény': teljesítmény, 'év': év}
 6.     return szótár
 7.
 8. def nagymestereket_listáz(nmk):
 9.     for nm in nmk:
10.         print(nm['név'], ': ', nm['teljesítmény'], ' (', nm['év'], ')', sep='')
11.
12. nagymesterek = [{'név': 'Polgár Judit',
13.                  'teljesítmény': 'Garri Kaszparov legyőzése',
14.                  'év': 2002},
15.                 {'név': 'Kempelen törökje',
16.                  'teljesítmény': 'Napóleon legyőzése (sakkban)',
17.                  'év': 1809},
18.                 {'név': 'Beth Harmon',
19.                  'teljesítmény': 'TV-sorozattá lett az "élete"',
20.                  'év': 2020}]
21.
22. nagymestereket_listáz(nagymesterek)
23.
24. bővítjük = None
25. while bővítjük != 'n':
26.     bővítjük = input('Akarsz új nagymestert megadni? (i/n) ')
27.     if bővítjük == 'i':
28.         nagymesterek.append(adatokat_bekér())
29.
30. print('A nagymesterek listája ilyen lett:')
31. nagymestereket_listáz(nagymesterek)
```

Vittük valamire – Típusalgoritmusok

A tizedik évfolyamos könyvünkben láttuk, hogy a típusalgoritmusokat – más néven programozási tételeket – azért érdemes ismerni, mert gyakran felmerülő problémákra adnak kipróbált megoldást. Az eddig megismert – úgynevezett „egyszerű” – típusalgoritmusok közös jellemzője, hogy egy bejárható objektumot vizsgálnak, és egy egyszerű értékre hivatkozva térnek vissza. Az egyszerű érték lehet például egy szám, a sorozat egy eleme vagy logikai érték: igaz vagy hamis.

Vannak olyan programozási nyelvek – ilyen a Python is –, ahol a típusalgoritmusoknak van rövid formájuk, függvényük is. A rövid formákat nagyon szeretjük (gyorsabban lehet beírni őket, könnyen olvashatók), de ebben a mostani könyvünkben is hangsúlyozzuk: sokszor olyan, összetettebb eset van, amikor csak a hosszabb, általánosabb forma használható. Fontos tehát, hogy a megoldás elvét is megismerjük. Így mindig pontosan az épp felmerülő problémának megfelelően tudjuk alkalmazni, kódolni a típusalgoritmusokat.

Sorozatszámítás

A sorozatszámítás algoritmusa arra jó, hogy egy bejárható objektum elemeit adjuk össze, szorozzuk össze, vagy írjuk egymás mellé. Azaz

ebből	ilyen művelettel	ilyet készít:
`[1,2,5]`	összeadás	`8`
`['ma', 'j', 'om']`	összefűzés	`'majom'`
`[2,3,4]`	átlagolás	`3`
`[1,2,3,4]`	összeszorzás	`24`

6. példa: Adjuk meg az év hosszát az egyes hónapok napjainak száma alapján!

```
1. hónapok_napjai = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
2.
3. #mindig használható megoldás
4. napok_száma = 0
5. for hónap in hónapok_napjai:
6.     napok_száma += hónap
7. print(napok_száma)
8.
9. #egyszerű megoldás:
10. napok_száma = sum(hónapok_napjai)
11. print(napok_száma)
```

Melyik fent említett művelethez tudunk még rövid formát adni?

Eldöntés

Az eldöntés algoritmusa arra a kérdésre ad választ, hogy van-e adott tulajdonságú elem a bejárható objektumunkban.

7. példa: Van-e 28 napos hónap az évben?

Az első megoldásunk feltételes ciklussal dolgozik, és így megfelel a *strukturált programozás* követelményeinek.

```
1. hónapok_napjai = [31,28,31,30,31,30,31,31,30,31,30,31]
2.
3. van_28 = False
4. index = 0
5. hossz = len(hónapok_napjai)
6. while index < hossz and not van_28:
7.     if hónapok_napjai[index] == 28:
8.         van_28 = True
9.     index += 1
10. if van_28:
11.     print('Van 28 napos hónap.')
12. else:
13.     print('Nem találtunk 28 napos hónapot.')
```

Itt tároljuk, hogy találtunk-e a keresett értékből.

A ciklus addig fut, amíg nem találunk olyan értéket, amilyet keresünk.

Ha találtunk, feljegyezzük.

Elágazás helyett használhatunk logikai kifejezést az értékadásra. A fenti kód hetedik sora ilyenkor a `van_28 = (hónapok_napjai[index] == 28)` formát ölti, cserébe nem kell a nyolcadik. A `hónapok_napjai[index] == 28` logikai kifejezés kiértékeléskor igaz vagy hamis lesz, és ezt az értéket kapja értékül a `van_28` változó.

A második megoldásunk bejárós ciklust használ. Most még kevéssé hatékony, mert bejárja az összes értéket. (A többi megoldást csak a harmadik sortól közöljük, lévén az első kettő változatlan.)

```
3. van_28 = False
4. for hónap in hónapok_napjai:
5.     if hónap == 28:
6.         van_28 = True
7. if van_28:
8.     print('Van 28 napos hónap.')
9. else:
10.     print('Nem találtunk 28 napos hónapot.')
```

A ciklus bejárja az összes értéket.

A továbbfejleszett változat már hatékony: ha talál a keresett értékből, akkor megszakítja a ciklust.

```
3. van_28 = False
4. for hónap in hónapok_napjai:
5.     if hónap == 28:
6.         van_28 = True
7.         break
8. if van_28:
9.     print('Van 28 napos hónap.')
10. else:
11.     print('Nem találtunk 28 napos hónapot.')
```

A ciklus bejárná az összes értéket.

Ha megvan, amit keresünk, félbehagyjuk a ciklust.

A Python nyelv úgy segíti elő a `break` utasítás használatát, azaz a ciklusokból való idő előtti kilépést, hogy a ciklusoknak létezik `else`-záradékuk – ez meglehetősen ritka dolog a programozási nyelvek körében. Az `else`-záradék csak akkor fut le, ha a ciklus végigfutott, azaz nem léptünk ki belőle `break`-kel. Ezt használjuk az utolsó megoldásunkban:

```
3. for hónap in hónapok_napjai:
4.     if hónap == 28:
5.         print('Van 28 napos hónap.')
6.         break
7. else:
8.     print('Nem találtunk 28 napos hónapot.')
```

Ez az else a for-hoz tartozik! Az else-ág kódja csak akkor fut le, ha nem volt break.

A fenti megoldások ugyanarra az eredményre vezetnek. Vannak, akik a bejárós ciklust használó megoldást könnyebben olvassák és értik, az elöltesztelő ciklust használó kód pedig bizonyítottan helyes megoldása a problémának.

Természetesen az eldöntésre is van egyszerű megoldás:

```
3. if 28 in hónapok_napjai:
4.     print('Van 28 napos hónap.')
```

Kiválasztás

A kiválasztás algoritmusa akkor használható, ha tudjuk, hogy van adott tulajdonságú elem a bejárható objektumunkban, és kíváncsiak vagyunk, hogy hányadik sorszámú ez az elem.

8. példa: Hányadik hónap a 28 napos?

Elsőként – a strukturált programozás követelményének eleget téve – elöltesztelő feltételes ciklust használunk.

A ciklus addig fut, amíg meg nem találjuk, amit keresünk.

```
1. hónapok_napjai = [31,28,31,30,31,30,31,31,30,31,30,31]
2.
3. index = 0
4. while hónapok_napjai[index] != 28:
5.     index += 1
6. print('A(z) ', index+1, '. hónap 28 napos.', sep='')
```

A kiválasztás is megvalósítható bejárós (számlálós) ciklussal. A kiírás kerülhet a ciklusba is, és utána is – írjuk oda, ahol az adott programban jobb helye van!

```
3. for index in range(len(hónapok_napjai)):
4.     if hónapok_napjai[index] == 28:
5.         break
6. print('A(z) ', index + 1, '. hónap 28 napos.', sep='')
```

Pythonban tudunk olyan bejárós ciklust írni, amelyik bejárja a kapott objektumot, és a bejárt értékeket számozza is. Ezt tekintjük pythonosabb megoldásnak:

két ciklusváltozó

enumerate = számozd be!

```
3. for index, nap in enumerate(hónapok_napjai):
4.     if nap == 28:
5.         break
6. print('A(z) ', index + 1, '. hónap 28 napos.', sep='')
```

És most is van egyszerű formánk:

```
3. print('A(z) ', hónapok_napjai.index(28)+1, '. hónap 28 napos.', sep='')
```

Keresés

A keresés algoritmusa az eldöntés és a kiválasztás egybeépítése. Van-e adott tulajdonságú elem, és ha igen, akkor hol?

9. példa: Ha van 28 napos hónap az évben, akkor hányadik hónap ilyen?

A megoldás feltételes ciklussal:

Ha nem találtunk 28 napos hónapot, akkor az index változó a lista utolsó elemén túlra fog mutatni, de ebből nem lesz baj, mert a while feltételben figyelünk rá.

```
1. hónapok_napjai = [31,28,31,30,31,30,31,31,30,31,30,31]
2.
3. index = 0
4. while index < len(hónapok_napjai) and hónapok_napjai[index] != 28:
5.     index += 1
6. if index < len(hónapok_napjai):
7.     print('A(z) ', index+1, '. hónap 28 napos.', sep='')
8. else:
9.     print('Nem találtunk 28 napos hónapot.')
```

Amikor nem találunk 28-at a listában, az index változó túlmutat a lista végén. Most nem mutat túl, azaz találtunk.

Bejárós ciklust használva:

```
3. hányadik_28 = None
4. for index in range(len(hónapok_napjai)):
5.     if hónapok_napjai[index] == 28:
6.         hányadik_28 = index
7.         break
8. if hányadik_28:
9.     print('A(z) ', hányadik_28 + 1, '. hónap 28 napos.', sep='')
10. else:
11.     print('Nem találtunk 28 napos hónapot.')
```

azaz:
if hányadik_28 != None

A következő megoldásban ismét két változóval járjuk be a ciklust, és a `for`-ciklushoz tartozó `else`-záradékot is használjuk:

```
3. for index, nap in enumerate(hónapok_napjai):
4.     if nap == 28:
5.         print('A(z) ', index + 1, '. hónap 28 napos.', sep='')
6.         break
7. else:
8.     print('Nem találtunk 28 napos hónapot.')
```

Az egyszerű megoldás természetesen az előző két típusalgoritmus egyszerű változatának egybeépítését jelenti:

```
3. if 28 in hónapok_napjai:
4.     print('A(z) ', hónapok_napjai.index(28)+1, '. hónap 28 napos.', sep='')
5. else:
6.     print('Nem találtunk 28 napos hónapot.')
```

Megszámolás

A megszámolás algoritmusa megmondja, hogy hány adott tulajdonságú elem van a bejárható objektumban.

10. példa: Hány 30 napos hónap van az évben?

```
1. hónapok_napjai = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
2.
3. #mindig használható megoldás
4. hány_30 = 0
5. for hónap in hónapok_napjai:
6.     if hónap == 30:
7.         hány_30 += 1
8. print(hány_30, '30 napos hónap van.')
9.
10. #egyszerű megoldás:
11. hány_30 = hónapok_napjai.count(30)
12. print(hány_30, '30 napos hónap van.')
```

Írhatunk most breaket?

Szélsőérték-kiválasztás: maximum- és a minimumkiválasztás

A maximum- és a minimumkiválasztás a legkisebb és a legnagyobb elemet keresi meg.

11. példa: Hány napos az év legrövidebb hónapja?

```
1. hónapok_napjai = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
2.
3. #mindig használható megoldás
4. legrövidebb = 32
5. for hónap in hónapok_napjai:
6.     if hónap < legrövidebb:
7.         legrövidebb = hónap
8. print('A legrövidebb hónap', legrövidebb, 'napos.')
9.
10. #egyszerű megoldás:
11. legrövidebb = min(hónapok_napjai)
12. print('A legrövidebb hónap', legrövidebb, 'napos.')
```

Ide olyan értéket adunk meg, ami biztos nem lehet a minimum. 32 napos hónap remélhetőleg nincs a listánkban.

Az előző példákban mindig használható volt a rövidebb, függvényes megoldási forma. Általános szabály, hogy a függvényes megoldás minden olyan esetben alkalmazható, amikor

pontosan azt az értéket akarjuk összeadni, megtalálni, megszámolni, ami a bejárható objektumban van. Ha nem az ott szereplő elemmel, hanem

- a belőle kiszámítható, meghatározható értékkel,
- vagy az elemek közül csak azokkal, amelyek megfelelnek valamilyen feltételnek,

tehát *nem mindegyikkel* kell dolgoznunk, az egyszerűbb, függvényes formák mit sem érnek.

Feladatok

A következő kérdések egy tíznapos periódus hajnali hőmérsékleteit tartalmazó listára vonatkoznak.

Keressünk három olyan kérdést, amely megoldható valamelyik típusalgoritmus egyszerűbb formájával, és három olyat, amely nem! Ez utóbbiak megoldását mindenképp kódoljuk is!

```
hőmérsékletek = [7, 5, -2, 0, -4, 3, 3, 3, 4, 4]
```

1. Hány fok volt a legmelegebb hajnalon?
2. Volt-e olyan hajnal, amikor fagyott? (A nullafokos víz vagy megfagyott már, vagy még éppen nem. Menjünk biztosra, nézzük a nulla foknál hidegebb hajnalokat!)
3. Volt-e olyan hajnal, amikor három fokot mértek?
4. Hány nullafokos hajnal volt?
5. Mikor volt először nullafokos hajnal?
6. Hányadik hajnalon volt fagy először?
7. Hányadik hajnalon volt utoljára fagy?
8. Hányszor fagyott hajnalban?
9. Volt-e olyan, hogy egymás utáni hajnalokon ugyanazt a hőmérsékletet mérték?
10. Hányadik hajnalon volt először melegebb, mint előző nap?

Az egyes kérdéseket megoldó programok megtalálhatók a tankönyv webhelyéről letöltött fájlok között.

Vittük valamire – Típusalgoritmusok kétdimenziós listákkal és szótárakkal

Amikor többdimenziós listákat vagy szótárakat vizsgálunk típusalgoritmusainkkal, csak egy-egy részfeladatban tudjuk használni a típusalgoritmusokat megoldó függvényeket. Azért van ez így, mert – ahogy már tizedik évfolyamon is láttuk – maguk az értékek nem állnak rendelkezésre azonnal feldolgozható formában.

Ennek a leckének az első részében az alábbi kétdimenziós listával dolgozunk:

```
1. hiányzók = [[3, 4, 0, 0, 1],
2.             [2, 3, 0, 0, 0],
3.             [4, 3, 2, 3, 4],
4.             [0, 0, 1, 0, 0]]
```

A nagy listában négy kisebb lista van.

12. példa: Hiányzók

A `hiányzók` lista egy négyhetes időszak egyes napjain mutatja a 11. zs osztály tanulóinak hiányzásait: például a második hét keddjén három fő hiányzott. A lista megadását a kódban az olvashatóság kedvéért egy üres sor követné, úgyhogy a következő kódok mind a 6. sortól kezdődnek majd.

1. Hány hiányzás volt összesen?

```
6. összes = 0
7. for hét in hiányzók:
8.     for nap in hét:
9.         összes += nap
```

A sorozatszámítást egymásba ágyazott ciklusok belsejében végezzük.

Ha szemfülesek vagyunk, észrevehetjük, hogy a belső ciklust kiválthatjuk az algoritmust megoldó függvénnyel.

```
6. összes = 0
7. for hét in hiányzók:
8.     összes += sum(hét)
```

2. Volt-e olyan hét, amikor nem volt hiányzó?

Ezúttal is hetenként járjuk be a listánkat. Mielőtt a bejárást megkezdenénk, ahogy az eldöntéskor szokás, kezdeti értékkel megadunk egy változót. A kezdeti érték megadásakor feltételezzük, hogy még nem volt hiányzásmentes hét, így az érték most hamis lesz, és ha találunk hiányzásmentes hetet, majd igazra állítjuk.

Ha már az imént szemfülesek voltunk, legyünk most is azok! Vegyük észre, hogy egy hét akkor hiányzásmentes, ha a heti hiányzások összege 0.

Megoldás a strukturált programozásnak megfelelő feltételes ciklussal:

```
index = 0
hossz = len(hiányzók)
while index < hossz and sum(hiányzók[index]) != 0:
    index += 1
if index < hossz:
    print('Volt hiányzásmentes hét.')
else:
    print('Nem volt hiányzásmentes hét.')
```

Sorozatszámítás az eldöntés belsejében!

Megoldás bejárós ciklussal:

```
for hét in hiányzók:
    if sum(hét) == 0:
        print('Volt hiányzásmentes hét.')
        break
else:
    print('Nem volt hiányzásmentes hét.')
```

3. Volt-e olyan hét, amikor ötnél kevesebb hiányzás volt?
Ugye látjuk, hogy egy relációs jel módosítása a lényegi változtatás az előző feladat kódjához képest?

4. Melyik héten volt a legtöbb hiányzás?
A feladat egyszerű lenne, ha rendelkezésre állnának a *heti* hiányzások számai. Nem állnak, de ki tiltja meg, hogy egy újabb listába kigyűjtsük magunknak őket?

```
heti_hiányzások = []

for hét in hiányzók:
    heti_hiányzások.append(sum(hét))

legtöbb = max(heti_hiányzások)
print('A(z) ', heti_hiányzások.index(legtöbb)+1, '. héten volt a legtöbb hiányzás.', sep='')
```

üres lista

Kigyűjtjük a heti hiányzásokat (összegzés).

maximumkiválasztás

keresés

A lecke elején azt mondtuk, ennyire összetett adatszerkezetekben kevés esetben használhatjuk majd a típusalgoritmusok egyszerű formáit, viszont a fenti hét sorban háromszor is használtunk egyet-egyet. Ez igaz, de mindhárom esetben egy egydimenziós listával dolgoztunk.

5. Hányadik héten volt egyetlen hiányzás?
Az egyik lehetséges megoldás az előző feladat kódjának enyhe módosítása. Ezúttal is kigyűjtjük a heti hiányzásokat, majd egyszerűen megkeressük, hogy hányadik indexű elem az 1. Így lényegében egyszerűbb a feladat, mint az előző – nincs benne maximumkiválasztás.

6. A hétfői vagy a keddi napokon hiányoztak-e többen a négy hét alatt?
Ez a feladat lényegében két, párhuzamosan is elvégezhető sorozatszámítás.

```
 6. hétfői = 0
 7. keddi = 0
 8.
 9. for hét in hiányzók:
10.     hétfői += hét[0]
11.     keddi += hét[1]
12.
13. if hétfői > keddi:
14.     print('Hétfői napokon hiányoznak többen.')
15. elif hétfői < keddi:
16.     print('Keddi napokon hiányoznak többen.')
17. else:
18.     print('Ugyanannyi a hiányzás a hétfői és keddi napokon.')
```

egyik sorozatszámítás

másik sorozatszámítás

13. példa: Kutyaoltás

A szótárakban tárolt objektumok esetében hasonló problémákkal kerülünk szembe, mint az előbb. Ha nagyon egyszerűen nézzük, kétféle esetben használunk szótárat. Az első eset az, amikor **egyetlen szótárat használunk sok azonos jellemző tárolására.** Ilyen eset például, hogy kinek hányas a dolgozata, melyik napon hányan néztek meg egy videót, vagy melyik napon hány kutyát oltott be az állatorvos. Ez utóbbi egy lehetséges szótára:

```
1. oltások = {'hétfő': 8, 'kedd': 14, 'szerda': 2, 'csütörtök': 3, 'péntek':13}
```

A szótár megadását a kódban az olvashatóság okán üres sor követné, így a következő kódok mind a harmadik sortól kezdődnek majd.

7. Hány kutyát oltott be a héten az állatorvos?

```
3. összes = 0
4. for szám in oltások.values():
5.     összes += szám
6. print('A héten', összes, 'kutya kapott oltást.')
```

A szótár értékeit járjuk be.

8. Volt-e olyan nap, amikor legalább tíz kutyát oltott be a doki?

```
3. volt = False
4. for szám in oltások.values():
5.     if szám >= 10:
6.         volt = True
7. if volt:
8.     print('Volt legalább tízkutyás nap.')
```

9. Melyik napon oltotta be a legkevesebb állatot az állatorvos?

```
3. legkevesebb_nap = None
4. legkevesebb_szám = 100000
5. for nap, szám in oltások.items():
6.     if szám < legkevesebb_szám:
7.         legkevesebb_nap = nap
8.         legkevesebb_szám = szám
```

A kulcsokat is bejárjuk, mert azokat kell kiírni.

```
8. print('A legkevesebb kutya ezen a napon kapott oltást:', legkevesebb_nap)
```

Vegyük észre, hogy a szótárak ilyen használata lényegében arra jó, hogy nevet kapjanak az egyes adatok. Használatuk egyéb tekintetben hasonló a listához.

A szótárak használatának másik gyakori esete, amikor nem azonos tulajdonságokat tárolunk. Ilyenkor **egy szótárba egy megfigyelt objektum kerül:** egy fa, egy zeneszám, egy csillag, egy ember. A szótár kulcsai a megfigyelt tulajdonságok: a fa magassága és faja, a csillag színe, távolsága, nagysága, az ember szempilláinak száma és az, hogy bal- vagy jobbkezes. Általában több objektumot tárolunk, és így lényegében egy szótárakból álló listával dolgozunk.

14. példa: Hazánk legmagasabb hegycsúcsai

A következő feladatok során egy olyan listával foglalkozunk, melynek elemei szótár típusúak. A szótárban található objektumok mindegyike országunk legmagasabb hegycsúcsait reprezentálja.

```
1. csúcsok = [{'név': 'Kékes', 'hegység': 'Mátra', 'magasság': 1014},
2.            {'név': 'Hidas-bérc', 'hegység': 'Mátra', 'magasság': 971},
3.            {'név': 'Galya-tető', 'hegység': 'Mátra', 'magasság': 964},
4.            {'név': 'Szilvási-kő', 'hegység': 'Bükk-vidék', 'magasság': 961},
5.            {'név': 'Péter hegyese', 'hegység': 'Mátra', 'magasság': 960},
6.            {'név': 'Kettős-bérc', 'hegység': 'Bükk-vidék', 'magasság': 958},
7.            {'név': 'Istállós-kő', 'hegység': 'Bükk-vidék', 'magasság': 958}]
```

A lista megadását a kódban az olvashatóság kedvéért megint csak egy üres sor követné, úgyhogy a következő kódok mind a 9. sortól kezdődnek majd.

1. Hány csúcs található a Bükk-vidéken?

```
 9. bükki_csúcsok = 0
10.
11. for csúcs in csúcsok:
12.     if csúcs['hegység'] == 'Bükk-vidék':
13.         bükki_csúcsok += 1
14.
15. print('A Bükk-vidék', bükki_csúcsok, 'csúcsa van a listában.')
```

megszámlálás

2. Mennyi a mátrai csúcsok magasságának átlaga?
Nem tudunk egyszerűen a `len()` függvénnyel darabszámot mondani, lévén a listának nem minden eleme számít. Meg kell számlálnunk azokat, amelyek számítanak, és csak azokat kell összegeznünk az átlaghoz.

```
 9. mátra_magasság_összesen = 0
10. mátra_darabszám = 0
11.
12. for csúcs in csúcsok:
13.     if csúcs['hegység'] == 'Mátra':
14.         mátra_magasság_összesen += csúcs['magasság']
15.         mátra_darabszám += 1
16.
17. mátra_átlag = mátra_magasság_összesen / mátra_darabszám
18. print('A mátrai csúcsok átlagosan', round(mátra_átlag), 'méteresek.')
```

sorozatszámítás

megszámlálás

3. Melyik a Bükk-vidék legmagasabb csúcsa?
Az itt közölt megoldásban külön változóban tároljuk az eddigi legmagasabb csúcs magasságát és nevét.

```
 9. bükk_magasság = 0
10. bükk_csúcs = None
11.
12. for csúcs in csúcsok:
13.     if csúcs['hegység'] == 'Bükk-vidék':
14.         if csúcs['magasság'] > bükk_magasság:
15.             bükk_magasság = csúcs['magasság']
16.             bükk_csúcs = csúcs['név']
17.
18. print('A Bükk-vidék legmagasabb csúcsa:', bükk_csúcs)
```

maximumkiválasztás

Másik megoldás lehet, hogy egyetlen változót, a legmagasabb bükki csúcs szótárát tároljuk. Kódoljuk ezt a megoldást is!

Bővítsük úgy a programunkat, hogy kérdezze meg a felhasználótól, hogy melyik hegységet vizsgálja! Ha ez már megy, akkor írja ki a programunk a kérdés elé a lehetséges választási lehetőségeket!

4. Van-e ezer méternél magasabb csúcs a listában?

```
 9. van = False
10.
11. for csúcs in csúcsok:
12.     if csúcs['magasság'] > 1000:
13.         van = True
14.         break
15.
16. if van:
17.     print('Van 1000 méternél magasabb csúcs.')
```

keresés

Ha *pontosan* értjük a kódot, és látjuk, hogy a kiértékelés során a `csúcs['magasság'] > 1000` helyére kerül a `True` vagy `False`, és a 16. sor ezt követően már az `if True` vagy `if False` formát ölti, akkor látjuk azt is, hogy a 11–14. sor átfogalmazható úgy, hogy az értékadáskor egy logikai kifejezést írunk:

```
11. for csúcs in csúcsok:
12.     van = csúcs['magasság'] > 1000
13.     if van:
14.         break
```

Akár az első, akár a második megfogalmazást választjuk, érdemes azt is észrevenni, hogy a kiírás esetleg elhelyezhető a ciklusban:

```
11. for csúcs in csúcsok:
12.     van = csúcs['magasság'] > 1000
13.     if van:
14.         print('Van 1000 méternél magasabb csúcs.')
15.         break
```

Bővítsük a programunkat: kérdezze meg a felhasználótól, hogy hány méteres magasságot vizsgáljon!

Fájlok a kérészéletű adatok helyett

A szemfülesebbeknek bizonyára feltűnt már, hogy létezik „néhány” program, amelyik háttértárról képes betölteni, illetve háttértárra képes menteni az adatokat, amelyekkel dolgozik. Mi eddig ilyen programot nem tudtunk írni, de épp itt az ideje ezen változtatni!

Ha elég messziről tekintünk a számítógépeken tárolt fájlokra, akkor alapvetően kétféle különböztetünk meg. Az egyikben szöveg van, méghozzá formázatlanul, olyan, amely csak számokat, betűket, szóközöket, írásjeleket tartalmaz. Nincs benne dőlt betű, aláhúzás, nincs megadva betűtípus és -méret, azaz ezek a fájlok szigorúan nem olyan szöveget tartalmaznak, mint amilyet irodai szövegszerkesztő alkalmazással készítünk. Az ilyen fájlok kiterjesztése nagyon gyakran `.txt`, az angol text, azaz ’szöveg’ szó alapján. A `.txt` kiterjesztésűeken kívül ebbe a csoportba tartozik még többek között a táblázatos adatokat tárolni képes `.csv` fájltípus, valamint a weboldalak kódját tartalmazó `.html` és `.css` kiterjesztéssel bíró állomány- és még jó néhány fájltípus. Szöveget tartalmazó fájlokat készíthetünk az úgynevezett egyszerű szerkesztőkkel, például a Windows Jegyzettömbjével, de ilyen fájlt ír minden IDE, az is, amivel a digitáliskultúra-órákon szerkesztjük programjainkat.

A „másik” fájltípus az összes többi. Kiterjesztéseik és a bennük tárolt adatok rendkívül változatosak. Idetartoznak a lefordított programok, a videók, a képek, a hangok, az irodai szövegszerkesztők és táblázatkezelők saját formátumú állományai, a tömörített fájlok, és még ki tudja, hányféle fájl.

Könyvünk programozásról szóló részében csak a szövegfájlokkal foglalkozunk. Ennek a leckénknek pedig minden példájában az alábbi, `allatok.txt` fájlt használjuk, amely megtalálható a tankönyv weblapjáról letöltött fájlok között. A fájlban kedvenc tanyánk háziállatai vannak felsorolva. Érdemes tudni, hogy a fájl tizenöt soros, és Latyak kacsa sora után nem kezdünk új sort, nem nyomunk Entert. Az egyes sorokban találjuk az állat nevét, azt, hogy miféle állatról van szó (a későbbiekben az egyszerűség kedvéért sokszor és pontatlanul fajtának nevezzük ezt a tulajdonságot), és azt is, hogy ez az állat hány éves.

```
Totyak kacsa 1
Lakli kutya 12
Hektor kutya 4
Puha birka 3
Nyeldekel kecske 5
Csinibaba szarvasmarha 3
Nyakas liba 2
Dinka malac 1
Coca malac 1
Smafu malac 3
Tarajos kakas 1
Csillag macska 2
Zokni macska 3
Pamacs birka 4
Latyak kacsa 2
```

Szövegfájlok beolvasása

Az első programunk mindössze annyit tesz, hogy megnyitja a fájlt, és kiírja a sorait, majd zárja a fájlt.

Figyeljük meg, hogy az `open()` függvény paraméteréül csak a fájl nevét adtuk meg. Ha a fájl nem abban a mappában van, ahonnan a programot futtatjuk, vagy ha elírjuk a nevet, vagy ha egyáltalán nincs is ilyen fájl, akkor a programunk panaszkodik miatta.

```
C:\Users\raerek\programok>beolvas_es_kiir.py
Traceback (most recent call last):
  File "C:\Users\raerek\programok\beolvas_es_kiir.py", line 1, in <module>
    forrásfájl = open('elírtuk.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'elírtuk.txt'
```

Bizonyára feltűnik az is, hogy amikor kiírja a programunk a fájl sorait – azaz sikeresen megtalálta a fájlt, megnyitotta, és olvasott belőle –, akkor is minden sor alatt egy üres sort is megjelenít. Miért van ez így?

Az ok az, hogy a szövegfájlok sorainak a végét egy Linuxon és macOS-en egy új sor, más néven *soremelés* (angolul: line feed, LF), Windowson pedig egy *kocsivissza* és egy *soremelés* karakter zárja. A *kocsi vissza* és a *soremelés* szavak az írógépek korából valók, és itt arra utalnak, hogy már a negyven-ötven évvel ezelőtt készült nyomtatók is a fájlok nyomtatásakor a *kocsi vissza* jel hatására vitték vissza a nyomtatófejet a sor elejére, és a *soremelés* karakter hatására igazították a lapot egy sorral lentebb.

A fájl sorainak végén tehát van egy vagy több olyan karakter, amelyekből tudni lehet, hogy új sor kezdődik. Ezt a karaktert a programunk lelkiismeretesen beolvassa és kiírja, azaz új sor kezdődik. Igen ám, de a `print()` függvény alapértelmezetten is új sort kezd minden sor kiírása után. Ez eredményezi az üres sorokat. Ha meg akarunk szabadulni tőlük, kényelmes lehet a `print()` függvény alapértelmezett sorvége jelét, az új sort író `'\n'` értéket felülírni egy üres karakterlánccal. Azaz a

```
print(sor, end='')
```

utasítást használni. A legtöbbször azonban nem kiírjuk, hanem tároljuk az adatainkat – például egy listában –, és ilyenkor a szövegfájl sorainak végén lévő jel, jelek folyamatos gondot jelentenek. Szerencsére a Pythonban létezik egy olyan tagfüggvény, amely a szövegek végén lévő, számunkra gondot okozó végződést eltávolítja. A tagfüggvény neve `strip()`, azaz: *lecsupaszít*. Próbáljuk a kódunk 3. sorát így átírni:

```
3. print(sor.strip())
```

Látjuk, hogy mostanra nincsenek üres soraink a kiíráskor. A `strip()` tagfüggvény elég ügyes: figyel arra, hogy CR karakter vagy CR és LF karakterpár zárja a sort, és azt veszi le, amit kell.

Ha a fájlban lévő adatokat tárolni szeretnénk, akkor egy nagyon hasonló programot írunk:

```
1. állatok = []
2.
3. forrásfájl = open('allatok.txt')
4. for sor in forrásfájl:
5.     sor = sor.strip()
6.     állatok.append(sor)
7. forrásfájl.close()
8.
9. print(állatok)
```

Üres lista, ebbe pakolunk majd.

Nyitjuk a fájlt.

Lecsupaszítjuk a sort.

A csupasz sort a lista végére fűzzük.

Zárjuk a fájlt.

Kiírjuk a listánkat.

A fájlnyitás, -feldolgozás, -zárás hármassága a legtöbb programozási nyelven a fentiekhez hasonlóan zajlik. A Python fájlműveletekhez hasonló másik, egyre elterjedtebb módszere az, hogy a fájlkezelést végző részt egy `with` utasítással kezdődő programrész belsejében helyezzük el. Ilyenkor a fájl zárására nincs szükség, mert a `with` belsejét elhagyva a Python automatikusan zárja a fájlt. A fenti program 3–7. sorait cseréljük az alábbi négyre:

```
with open('allatok.txt') as forrásfájl:
    for sor in forrásfájl:
        sor = sor.strip()
        állatok.append(sor)
```

A with belsejében vagyunk, itt nyitva van a fájl.

Használjuk a két módszer közül azt, amelyik jobban kézre áll.

Sorokat már be tudunk olvasni, de a legtöbbször szeretnénk a bennük lévő adatokat szétszedve látni – a mostani példában külön az állat nevét, fajtáját és korát. Listák szétdarabolására, hasítására a `split()` tagfüggvény való. A működését az alábbi program szemlélteti. Próbáljuk ki a programot!

```
1. mondat = input('Írj ide egy mondatot! ')
2. szavak = mondat.split()
3. print('A szavakat tartalmazó lista:', szavak)
4. print('Újra összerakva:')
5. print(' '.join(szavak))
```

A split listává alakítja a karakterláncot.

A `split()` tagfüggvény alapértelmezetten szóközöknél, tabulátoroknál és hasonló karaktereknél (angol összefoglaló nevük: whitespace) hasít. Ha nekünk például pontosvessző jelzi az egyes adatok határát, akkor a `split(';')` formában használjuk az utasítást.

Az állatos programunkat az 5. sorban bővítsük a sor végére írt `split()` tagfüggvénnyel! Futtassuk a programot, és figyeljük meg a kimenetét: egy kétdimenziós listát látunk.

```
1. állatok = []
2.
3. forrásfájl = open('allatok.txt')
4. for sor in forrásfájl:
5.     sor = sor.strip().split()
6.     állatok.append(sor)
7. forrásfájl.close()
8.
9. print(állatok)
```

A sor karakterláncot először lecsupaszítjuk, majd darabjaira szedjük. Az eredmény egy lista, amelyet visszateszünk a sor nevű változóba.

A sor nevű listát az „állatok" nevű lista végére tesszük. Az „állatok" egy kétdimenziós lista.

Figyeljük meg, hogy a beolvasott számokat – az állatok korát – egyelőre szövegként tároljuk. Legcélszerűbb talán még a beolvasás során számmá alakítanunk őket. Ezt megtehetjük a fenti kód 5. és 6. sora közé szúrt

```
sor[2] = int(sor[2])
```

utasítással.

Természetesen semmi nem kötelez bennünket arra, hogy kétdimenziós listába töltsük az adatainkat. Ha a fenti kód 5. és 6. sora közé ezt az utasítást szúrjuk:

```
szótár = {'név': sor[0], 'fajta': sor[1], 'kor': int(sor[2])}
```

illetve az *állatok* lista végére nem a sorlistát, hanem a belőle képzett `szótár` nevű szótárat illesztjük a jelenlegi 6., a módosítást követően 7. sorba, akkor meg is oldottuk a feladatot.

A típusalgoritmusainkat természetesen a fájlokból beolvasott adatokkal is tudjuk használni. Ha például a beolvasást követően ki szeretnénk írni a legöregebb állat nevét és fajtáját, akkor a szótárakat használó változatot nagyjából hasonlóan kell kiegészítenünk:

A „legöregebb" az egész szótárat tárolja, nem csak az állat korát. Kezdetben a legelső állatot tekintjük a legöregebbnek.

```
12. legöregebb = állatok[0]
13. for állat in állatok:
14.     if állat['kor'] > legöregebb['kor']:
15.         legöregebb = állat
16.
17. print(legöregebb['név'], legöregebb['fajta'])
```

Ha találunk öregebb állatot, akkor tároljuk.

Szövegfájlok írása

A szövegfájlok írása nem sokkal bonyolultabb az olvasásuknál. A fájl megnyitását ezúttal is az `open()` függvény végzi. A függvény elhagyható paramétere a megnyitás módját jelzi. Ennek alapértelmezett értéke az `'r'`, azaz olvas (angolul: read), és ezt eddig nem írtuk ki soha, mert olvasni akartunk a fájlból. Ha írni akarunk egy fájlba, akkor az `open()` függvény a `'w'` (write, azaz ír) paraméter hatására új fájlt kezd a megadott néven – ha volt már ilyen, azt felülírja. Az `'a'` (append: hozzáfűz) paramétert megadva pedig az esetleg meglévő fájlt folytathatjuk.

Megnyitottuk hát a fájlunkat, de még írni is kell bele. Mindössze annyi a dolgunk, hogy a `print()` függvénynek átadjuk azt a fájlobjektumot, amibe írni kell.

Módosítsuk úgy az előző programunkat, hogy a legöregebb állat nevét ne a képernyőre, hanem az `oreg.txt` állományba írja! Mindössze az előző programunk utolsó sorát kell egy kicsit kiegészítenünk, illetve elé és mögé beírni egy-egy újat:

```
17. célfájl = open('oreg.txt', 'w')
18. print(legöregebb['név'], legöregebb['fajta'], file=célfájl)
19. célfájl.close()
```

Írásra nyitjuk meg a fájlt.

Ebbe a fájlba írjon a print().

Az írásra megnyitott fájlt mindenképp zárjuk be!

Most, hogy már úgy írunk és olvasunk fájlokat, mintha mindig is remekül ment volna, ki kell térnünk egy fontos részletre. A számítógép a karaktereket számkóddal tárolja, mind a memóriában, mind a fájlokban. A magyar nyelv ékezetes karaktereinek számmá alakítására sokféle módszer van. Az átalakítás mostanra jó pár éve szinte mindenhol elterjedt szabványa az UTF-8. 2021-ben, e sorok születése idején egyes statisztikák szerint a világháló tartalmának 97 százaléka így van kódolva. A mai Linuxok és macOS-ek is ezt használják.

Nos, a Windows parancssora kivétel, és a Windowson futó Python ezt követi. Ha azt szeretnénk, hogy az ékezetes karaktereket tartalmazó, általában UTF-8 kódolással mentett szövegfájljainkat a Pythonban írt programjaink Windowson hiba nélkül olvassák és írják, akkor tennünk kell ennek érdekében.

A helyzetet megoldja, ha a PYTHONUTF8 környezeti változót 1-re állítjuk. Ezt legkönnyebben a parancssorban kiadott `set PYTHONUTF8=1` paranccsal tehetjük meg. A módszer hátránya, hogy minden parancssori ablak megnyitásakor újra ki kell adni a parancsot. A környezeti változók tartósabb megadására szolgáló módszert találhatunk az interneten.

A másik megoldás, ha programjainkban az `open()` függvény argumentumai között szerepeltetjük, hogy `encoding='utf-8'`, íráskor és olvasáskor egyaránt. Így a programjaink Windowson gond nélkül lefutnak, Linuxon és macOS-en pedig nem lesz változás, azaz ott is gond nélkül lefutnak.

Minderre nincs szükség, ha a szövegfájlunk nem tartalmaz ékezetes karaktert, épp ezért nincs ékezet az `allatok.txt` fájlban lévő állatok nevében és fajtájában sem.

A szövegfájlok beolvasásának egyéb módszerei

A szövegfájlok soronkénti beolvasása az esetek nagy többségében megfelel a munkánkhoz. Ha mégsem így volna, az alábbiakban közöljük még a szövegfájlok beolvasásának néhány egyéb módszerét.

Beolvasás egy lépésben egy listába: a lista sorainak a végén ott vannak a sorvége jelek is, amelyeket nekünk kell leszedni utólag:

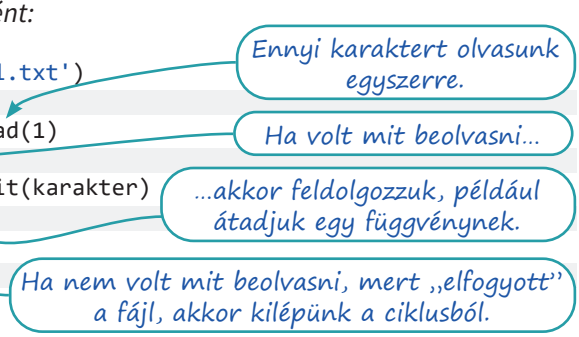
```
1. forrásfájl = open('szovegfajl.txt')
2. lista = forrásfájl.readlines()
3. forrásfájl.close()
```

Beolvasás egy lépésben egyetlen szövegként: ritkán használjuk, de az ezt követő módszer megértéséhez szükség van rá:

```
1. forrásfájl = open('szovegfajl.txt')
2. szöveg = forrásfájl.read()
3. forrásfájl.close()
```

Beolvasás egy vagy néhány karakterenként:

```
1. forrásfájl = open('szovegfajl.txt')
2. while True:
3.     karakter = forrásfájl.read(1)
4.     if karakter:
5.         csinálunk_vele_valamit(karakter)
6.     else:
7.         break
8. forrásfájl.close()
```



Kópiakészítés és digitális Hamupipőke – Másolunk, kiválogatunk és szétválogatunk típusalgoritmussal

Tizedik évfolyamon megismerkedtünk a típusalgoritmusok – ha úgy tetszik, programozási tételek – egy csoportjával, mostanra pedig fel is elevenítettük használatukat. Az eddig megismertek az „egyszerű” típusalgoritmusok. Közös tulajdonságuk, hogy egy bejárható objektum sok eleméhez egyetlen értéket – az összegüket, az átlagukat, a legnagyobbat, egy kiválasztott elem értékét, egy logikai értéket – rendelünk velük.

E mostani leckével indulóan olyan típusalgoritmusokat ismerünk meg, amelyek egy értéksorozathoz, bejárható objektumhoz egy másik bejárható objektumot rendelnek. Ezek az **összetett típusalgoritmusok.**

Másolás

A másolás típusalgoritmusának lényege a következő:

1. járjuk be egy bejárható objektum elemeit,
2. valamilyen módon alakítsuk át az egyes elemeket, és az átalakítást követően
3. gyűjtsük újabb bejárható objektumba az így kapott elemeket.

Az elemek átalakítását matematikusmegfogalmazással mondhatjuk úgy is, hogy a bejárható objektum elemeihez valamilyen hozzárendelési szabállyal, hozzárendelő függvénynyel másik elemet rendelünk.

Mindez mondatszerű leírásban:

```
sorozat = valamilyen bejárható objektum
másolat = üres bejárható objektum
ciklus a sorozat minden elem-ére
    átalakított_elem = hozzárendelő_függvény(elem)
    másolat-hoz hozzáfűz(átalakított_elem)
ciklus vége
```

Megjegyezzük, hogy amikor a „hozzárendelő függvény” feladata kellőképp egyszerű, nem feltétlen írunk külön függvényt, hanem helyben kiszámítjuk a hozzárendelés értékét.

Az egyik leggyakoribb másolási feladat a nagyon gyakran előforduló típusátalakítás, például amikor karakterekként tárolt számokat alakítunk „igazi” (egész vagy lebegőpontos) számokká, vagy fordítva. Három módszert is bemutatunk.

Az első a leghagyományosabb, teljesen megfelel a fenti mondatszerű leírásnak.

A másodikban szereplő `map()` függvénynek két paramétere van: az első egy függvény, a második egy bejárható objektum. A `map()` a paraméterként megadott függvényt a bejárható objektum minden elemén lefuttatja. A `map()` visszatérési értékét a `list()` függvénnyel listává alakítjuk. Azért közöljük ezt a módszert, mert Python nyelven ez a legrövidebb megfogalmazása a másolás algoritmusának.

A harmadik módszer a listaértelmezés (angolul: list comprehension) módszere. Lényegében ugyanaz, mint az első, csak egy sorba átírva. Python nyelvű kódokat keresgélve nagyon gyakran találkozunk ezzel a módszerrel az interneten.

```
1. számok_string_formában = ['1', '5', '2', '3', '4']
2. print('Stringként: ', számok_string_formában)
3.
4. # első módszer: ciklus
5. számok = []
6. for string in számok_string_formában:
7.     szám = int(string)
8.     számok.append(szám)
9. print('Első módszerrel: ', számok)
10.
11. # második módszer: map
12. számok = list(map(int, számok_string_formában))
13. print('Második módszerrel: ', számok)
14.
15. # harmadik módszer: listaértelmezés
16. számok = [ int(string) for string in számok_string_formában ]
17. print('Harmadik módszerrel: ', számok)
```

Most az int() függvény a mondatszerű leírásban emlegetett „hozzárendelő" függvény.

A map() az itt megadott függvényt futtatja le...

...ennek az objektumnak minden elemén.

Ez a két forma ugyanazt csinálja.

15. példa: A taxis bevételei

Adott a tizedik évfolyamos tankönyvből ismerős taxisunk piculában kifejezett bevételeinek listája. A taxis adóterhei jelentősek: mire kifizeti a mindenféle adókat, a bevétel egészre kerekített 49 százalékáról lemondhat. Adjuk meg a taxis adózott bevételeinek a listáját!

```
1. bevételek = [1, 5, 2, 3, 4]
2. adózott_bevételek = []
3.
4. for bevétel in bevételek:
5.     adó = round(bevétel * 0.49)
6.     marad = bevétel - adó
7.     adózott_bevételek.append(marad)
8.
9. print('A taxis adózott bevételei:', ', '.join(map(str, adózott_bevételek)))
```

Most a round() és a szorzás a „hozzárendelő" függvény.

Itt meg valójában végzünk egy másik másolást.

16. példa: Libatömegek farkas előtt és farkas után

Az a szituáció is ismerős lehet a tizedik évfolyamos kötetből, amikor a róka libát lop a faluból. A libák súlyát – pontosabban tömegét – listában adjuk meg. A farkas a dűlőútnál várja a rókát, és a három kilónál nagyobb libákat elveszi – a piciket nagylelkűen otthagyja a rókának. Adjuk meg azt a listát, amelyik a farkassal való találkozást követő libatömegeket tartalmazza! Amelyik libát a farkas elvette, annak helyére írjunk nullát!

```
1. def farkas(liba_tömege):
2.     if liba_tömege > 3:
3.         return 0
4.     else:
5.         return liba_tömege
6.
7. libák = [1, 5, 2, 3, 4]
8. róka_libái = []
```

```
 9.
10. for liba in libák:
11.     marad = farkas(liba)
12.     róka_libái.append(marad)
13.
14. print('A rókánál maradt libák:', ', '.join(map(str, róka_libái)))
```

17. példa: A tanya állathangjai

Az előző leckéből már ismert `allatok.txt` fájl tartalmazza tanyánk állatait. Minden állatfajnak ismert a hangja (például macska: nyaú, malac: röf). Állítsuk össze azt a listát, amely megmutatja, hogy milyen hangokkal köszöntenek bennünket állataink, amikor hazaérünk a tanyára! Feltételezzük, hogy az állatok a fájlban található sorrendjüknek megfelelően szólalnak meg.

```
 1. állathangok = {'kacsa': 'háp', 'kutya': 'vaú',
 2.                'birka': 'bee', 'kecske': 'mek',
 3.                'szarvasmarha': 'mú', 'liba': 'gá',
 4.                'malac': 'röf', 'kakas': 'qqriq',
 5.                'macska': 'nyaú'}
 6.
 7. köszöntések = []
 8.
 9. forrásfájl = open('allatok.txt')
10. for sor in forrásfájl:
11.     fajta = sor.strip().split()[1]
12.     köszöntések.append(állathangok[fajta])
13.
14. forrásfájl.close()
15.
16. print('Állataink üdvözlete: ', ', '.join(köszöntések))
```

Kiválogatás és szétválogatás

A kiválogatás típusalgoritmusa majdnem olyan, mint a másolásé. Ezúttal nem alakítjuk át az elemeket, hanem az eredetieket másoljuk, de nem mindet, hanem csak azokat, amelyek megfelelnek valamilyen feltételnek.

Mindez mondatszerű leírásban:

```
sorozat = valamilyen bejárható objektum
kiválasztottak = üres bejárható objektum
ciklus a sorozat minden elem-ére
   ha elem adott tulajdonságú, akkor:
      kiválasztottak-hoz hozzáfűz(elem)
ciklus vége
```

A szétválogatás típusalgoritmusa abban különbözik a kiválogatásétól, hogy több listába vagy egyéb objektumba válogatjuk szét az elemeket. Az egyikbe kerülnek azok, amelyek megfeleltek a feltételnek, a másikba azok, amelyek nem. Az is elképzelhető, hogy többfelé

válogatjuk szét az eredeti elemsorozatunkat: az egyik gyűjteménybe kerülnek a kicsik, a másikba a közepesek, a harmadikba a nagyok.

18. példa: A farkas és a róka libalakomája

Az 16. példában szereplő libatolvajlások ismeretében adjuk meg a két ragadozó szárnyasainak listáit! A listák felhasználásával állapítsuk meg, hogy melyik ragadozónak hány darab, illetve hány kilónyi liba jutott! A kiírást végeztessük eljárással, melynek paraméterei a ragadozó neve és a ragadozó tyúkjait tartalmazó lista!

```
1. def kiír(ragadozó, libalista):
2.     print('A', ragadozó, 'libái:',
3.         ', '.join(map(str, libalista)))
4.     print('A libák száma:', len(libalista),
5.         'össztömege:', sum(libalista), 'kg.')
6.
7. libák = [1, 5, 2, 3, 4]
8. róka_libái = []
9. farkas_libái = []
10.
11. for liba in libák:
12.     if liba <= 3:
13.         róka_libái.append(liba)
14.     else:
15.         farkas_libái.append(liba)
16.
17. kiír('róka', róka_libái)
18. kiír('farkas', farkas_libái)
```

A join() csak stringeket tud összefűzni, szóval konvertálunk.

megszámlálás

sorozatszámítás/összegzés

Szétválogatás: a nagy libák a farkasé, a kicsik a rókáé.

Az eljárást kétszer hívjuk.

19. példa: Hőségriadós napok

A hőségriasztás legalacsonyabb fokozata arról tájékoztat, hogy egy napon 25 °C-ot meghaladó hőmérséklet várható. A következő heti előrejelzés szótárban tárolt adatai alapján gyűjtsük listába, és írjuk a képernyőre azokat a napokat, amikor hőségriasztást kell kiadni!

```
1. előrejelzések = {'hétfő': 19, 'kedd': 23, 'szerda': 26,
2.                  'csütörtök': 27, 'péntek': 19,
3.                  'szombat': 18, 'vasárnap': 18}
4.
5. hőségriadós_napok = []
6. for nap in előrejelzések:
7.     if előrejelzések[nap] > 25:
8.         hőségriadós_napok.append(nap)
9.
10. print('Hőségriadós napok:', ', '.join(hőségriadós_napok))
```

A szótárat kulcsok szerint járjuk be. Írhatnánk azt is, hogy előrejelzések.keys().

kiválogatás

Kópiakészítés és digitális Hamupipőke – Az eddig tanult összetett típusalgoritmusok a gyakorlatban

20. példa: Gyalogtúra

A könyv webhelyéről letöltött `tura.txt` állományban egy gyalogtúrán hárompercenként rögzített magassági adatokat találunk. Olvassuk be a fájlt, majd állítsunk elő a felhasználásával egy olyan listát, amelyik azt mutatja, hogy „Fel” vagy „Le” változott-e a túrázónál lévő GPS-készülék által mért magasság az előző mérési pont óta, esetleg megegyezik az előzővel („=”)! Írjuk az új lista elemeit vesszővel elválasztva a képernyőre!

Írjuk meg a fájlbeolvasást követő részt mondatszerű leírással! Minthogy (majdnem) minden adatnak megfeleltetünk egy újat, a másolás típusalgoritmusát kell használnunk.

```
Program szintmérés:
magasságok := fájlból beolvasott adatok bejárható objektumban
irányok := üres bejárható objektum
ciklus i = 1-től (magasságok elemszáma)-1 -ig
   ha magasságok[i] < magasságok[i-1], akkor
     irányok := irányok + „Le”
   különbenha magasságok[i] > magasságok[i-1], akkor
     irányok := irányok + „Fel”
   különben
     irányok := irányok + „=”
ciklus vége
Program vége.
```

Kódoljuk a programot, és mentsük `szintmeres.py` néven!

```
1. with open('tura.txt') as forrásfájl:
2.     magasságok = forrásfájl.read().strip().split(', ')
3.
4. magasságok = list(map(int, magasságok))
5.
6. irányok = []
7. for index in range(1, len(magasságok)):
8.     if magasságok[index] < magasságok[index-1]:
9.         irányok.append('Le')
10.    elif magasságok[index] > magasságok[index-1]:
11.        irányok.append('Fel')
12.    else:
13.        irányok.append('=')
14.
15. print('A magasság változása:', ', '.join(irányok))
```

Megnyitjuk a fájlt.

Beolvassuk az első (és egyetlen) sorát.

Levesszük a sorvége jel(ek)et.

Vessző és szóköz mentén daraboljuk a sort.

Egésszé alakítjuk a „magasságok” lista minden egyes értékét.

Megvalósítjuk a fenti pszeudokódot.

Módosítsuk az előző programot úgy, hogy a három métert meg nem haladó változásokat még egyenlőnek tekintse! Mindössze a fenti kód 8. és 10. sorában kell egy keveset változtatnunk: a kettőspont elé kell odaírni, hogy „-3”, illetve „+3”.

21. példa: E terkes mecske leberetette e tejfelt

Lehet egy egyperces rettenet, melyet eszperente nyelven egy ember nyelvel? Az `eszperente.py` program egy hangyányit primitív lesz, ugyanis a megadott mondatból úgy ír eszperentét, hogy minden magánhangzót *e*-re cserél, például:

A torkos macska leborította a tejfölt. *E terkes mecske leberetette e tejfelt.*

Az első verzió elég, ha csupa kisbetűs mondatokat kezel, aztán oldjuk meg, hogy a nagybetűket is tudja kezelni! Minthogy minden karakternek megfeleltetünk egy másikat, ismét a másolás típusalgoritmusa fog segíteni.

```
1. magyar_mondat = input('Kérek egy mondatot! ')
2. eszperente_mondat = []
3.
4. magánhangzók = 'aáeéiíoóöőuúüű'
5.
6. for karakter in magyar_mondat:
7.     if karakter in magánhangzók:
8.         eszperente_mondat.append('e')
9.     else:
10.        eszperente_mondat.append(karakter)
11.
12. print('Gyenge eszperente:', ''.join(eszperente_mondat))
```

Be kell-e ide írni az „e” betűt?

Eldöntés a másolás kellős közepén. Mit dönt el?

A nagybetűket is kezelő változat:

```
6. for karakter in magyar_mondat:
7.     if karakter.lower() in magánhangzók:
8.         if karakter.isupper():
9.             eszperente_mondat.append('E')
10.        else:
11.            eszperente_mondat.append('e')
12.    else:
13.        eszperente_mondat.append(karakter)
```

A kisbetűs változat alapján döntünk.

Nagy magánhangzó esetén nagy E kerül az eszperente változatba…

…kicsi esetén kis e.

22. példa: Kutya- és macskaoltások

Kutyáinkat és macskáinkat be szeretnénk oltatni. Mindegyiknek adatnánk veszettség elleni oltást, és a kutyáknak parvovírus ellenit is. A már használt `allatok.txt` fájlból gyűjtsük az oltás nevének megfelelő listába azoknak az állatoknak a nevét, amelyek az adott oltást kapni fogják! Jelenítsük meg a listák tartalmát!

Ki kell válogatnunk, hogy mely állatok tartoznak az egyik, illetve a másik fajhoz. Ehhez a kiválogatás típusalgoritmusát használjuk.

Írjuk meg mondatszerű leírással a két oltási listát kialakító részt! Feltételezhetjük, hogy az állatokat kétdimenziós listában tároltuk a fájl beolvasását követően.

```
Program kutyamacska:
veszettség := üres gyűjteményes adatszerkezet
parvo := üres gyűjteményes adatszerkezet
ciklus állatok minden állat-jára
  ha állat[1] = „kutya” vagy állat[1] = „macska” akkor
```

```
    veszettség-et bővítjük állat[0]-val
  ha állat[1] = „kutya” akkor
    parvo-t bővítjük állat[0]-val
ciklus vége
Program vége.
```

Kódoljuk a programot `kutyamacska.py` nevű állományba!

```
 1. állatok = []
 2.
 3. with open('allatok.txt') as forrásfájl:
 4.     for sor in forrásfájl:
 5.         sor = sor.strip().split()
 6.         sor[-1] = int(sor[-1])
 7.         állatok.append(sor)
 8.
 9. veszettség = []
10. parvo = []
11.
12. for állat in állatok:
13.     if állat[1] == 'kutya' or állat[1] == 'macska':
14.         veszettség.append(állat[0])
15.     if állat[1] == 'kutya':
16.         parvo.append(állat[0])
17.
18. print('Veszettség ellen kap oltást:', ', '.join(veszettség))
19. print('Parvovírus ellen kap oltást:', ', '.join(parvo))
```

Az utolsó adat szám, számként is tároljuk. Még akkor is rendesen tároljuk az adatainkat, ha most épp nem lesz rájuk szükség.

23. példa: Tojásrakók

Van egy listánk arról, hogy a háziállataink közül melyik faj egyedei raknak tojást. A már használt `allatok.txt` fájlból* gyűjtsük ki azoknak a nevét, amelyeknél elképzelhető ilyen esemény! Az állatok nevéből ezúttal ne következtessünk a nemükre!

Ki kell válogatnunk a tojásrakókat – ezt a típusalgoritmust használjuk a `tojasrakok.py` programban.

A kód eleje megegyezhet az előző feladat megoldásáéval, így csak a 9. sortól kezdődő részt adjuk meg itt.

```
 9. tojásrakó_fajok = ['kacsa', 'liba', 'kakas']
10. tojásrakók_nevei = []
11.
12. for állat in állatok:
13.     if állat[1] in tojásrakó_fajok:
14.         tojásrakók_nevei.append(állat[0])
15.
16. print('Tojásrakó fajhoz tartozik:', ', '.join(tojásrakók_nevei))
```

*Tarajos fajként a fájlunkban „kakas” szerepel, aminek az az oka, hogy a „házi tyúk” szóban szerepel ékezetes karakter, és ez – mint ahogy láttuk – bonyolíthatja a programot.

Kópiakészítés és digitális Hamupipőke – Az eddig tanult összetett típusalgoritmusok újabb gyakorlatokban

24. példa: Jók és rosszak

George Orwell *Állatfarm* című regényében egy tanya állatai fellázadnak gonosz gazdájuk, illetve általában az emberek ellen. A legegyszerűbb gondolkodásúak számára is világossá óhajtották tenni az új világrendet, így a lázadás vezetői kiadták a „Négy láb jó, két láb rossz!” jelszót. A szárnyasok rámutattak, hogy ez a szlogen számukra kirekesztő, mire a lázadás ideológusai elmagyarázták, hogy ebben a kontextusban a madarak szárnya is lábnak minősül. Nincs hát szó a szárnyasok megbélyegzéséről, a jelszó az emberi lényeket minősíti.

Írjunk programot az előző leckében megírt program átalakításával `orwell.py` néven, amely a jelszónak még az említett utólagos korrekciója előtt, a jelszó szigorúan vett jelentése szerint kategorizálja állatainkat! Olvassuk be soronként az `allatok.txt` fájlt, és minden egyes sor beolvasását követően helyezzük a `jók`, illetve a `rosszak` listába a sorban szereplő állat nevét az előző feladatban is segítségünkre lévő lista felhasználásával! Jelenítsük meg vesszővel elválaszott felsorolásként az egyes listák tagjait!

A program minden állatot megtart, elhelyez egy gyűjteményes adatszerkezetben, azaz a szétválogatás típusalgoritmusát használjuk.

```
állatok = []

with open('allatok.txt') as forrásfájl:
    for sor in forrásfájl:
        sor = sor.strip().split()
        sor[-1] = int(sor[-1])
        állatok.append(sor)

tojásrakó_fajok = ['kacsa', 'liba', 'kakas']
jók = []
rosszak = []
for állat in állatok:
    if állat[1] in tojásrakó_fajok:
        rosszak.append(állat[0])
    else:
        jók.append(állat[0])

print('Rosszak:', ', '.join(rosszak))
print('Jók:', ', '.join(jók))
```

szétválogatás

Ha elkészültünk, bővítsük a programot úgy, hogy az egyes sorok beolvasását követően adjon szövegesen megfogalmazott véleményt is, például:

```
Totyak kacsa kétlábú, azaz rossz.
```

Úgy látjuk, hogy a véleményalkotást és a vélemény megjelenítését már megéri függvénybe kiszerveznünk. Írjunk hát olyan függvényt,

- melynek paramétere egy szótár, amely a függvénynek átadott állat nevét, faját és korát tartalmazza;
- amelynek helyi, csak a függvény belsejében létező adatszerkezete a tojásrakó fajokat felsoroló, a döntéshozatal alapjául szolgáló lista;
- amelyik a fenti mintának megfelelő véleményezést ír a képernyőre és
- amelyik visszatérési értéke az adott állat „jóságát” jelentő logikai érték!

A `jók` és a `rosszak` listát a programunk új változata az előbb elkészített függvény használatával töltse fel!

```
 1. def jó(állat):
 2.     tojásrakó_fajok = ['kacsa', 'liba', 'kakas']
 3.     if állat['faj'] in tojásrakó_fajok:
 4.         print(állat['név'], állat['faj'], 'kétlábú, tehát rossz.')
 5.         return False
 6.     else:
 7.         print(állat['név'], állat['faj'], 'négylábú, tehát jó.')
 8.         return True
 9.
10. állatok = []
11.
12. with open('allatok.txt') as forrásfájl:
13.     for sor in forrásfájl:
14.         sor = sor.strip().split()
15.         állat = {'név': sor[0], 'faj': sor[1], 'kor': int(sor[2])}
16.         állatok.append(állat)
17.
18. jók = []
19. rosszak = []
20. for állat in állatok:
21.     if jó(állat):
22.         jók.append(állat['név'])
23.     else:
24.         rosszak.append(állat['név'])
25.
26. print('Rosszak:', ', '.join(rosszak))
27. print('Jók:', ', '.join(jók))
```

egy állat = egy szótár

Itt hívjuk a függvényt.

A szétválogatás a függvény visszatérési értéke alapján történik.

25. példa: Hajónapló

Adott egy fájl `hajonaplo.txt` néven, melyet a tankönyv weblapjáról letöltött fájlok között találunk. A fájl soronként két, kötőjellel elválasztott számot tartalmaz. Az első szám minden sorban azt mutatja, hogy hány fokos irányba haladt a hajó, a második azt, hogy hány tengeri mérföldet haladt abba az irányba. A `kormanyos.py` programban az a feladatunk, hogy megadjunk egy olyan listát, amely azt sorolja fel, hogy a hajót az egyes irányváltások – az egyes sorok – között jobbra vagy balra kormányozták-e. Mindig arra kormányozzák a hajót, amerre kisebbet kell fordulnia. Ha pontosan száznyolcvan fokot kellene fordulni, akkor a kormányos véletlenszerűen dönti el, hogy jobbra vagy balra fordul-e.

Majdnem minden adatnak megfeleltetünk egy újat, azaz a másolás típusalgoritmusát kell használnunk.

A fordulás irányát meghatározó algoritmus lehet például a következő:

```
ha (új_irány - régi_irány + 360) mod 360 < 180 akkor
    ki: „J”
különbenha (új_irány - régi_irány + 360) mod 360 > 180, akkor
    ki: „B”
különben:
    ki: „J” és „B” közül valamelyik véletlenszerűen
```

Írjunk a fenti mondatszerű leírásból függvényt, amelynek két paramétere a régi és az új irány, a visszatérési értéke pedig egy J vagy egy B karakter! Írjuk meg a főprogramot, amely beolvassa a `hajonaplo.txt` fájlt, és a kormányzásokat a `kormanyzasok.txt` fájlba írja! A kimeneti fájl egy-egy sora tartalmazza a kormánymozdulat előtti útirányt, a kormánymozdulat irányának betűjét és a kormánymozdulatot követő irányt, szóközzel elválasztva. Ha a bemeneti fájl adatai:

```
107-12
109-34
210-87
202-3
349-198
17-251
197-37
```

akkor a kimeneti fájléi:

```
107 J 109
109 J 210
210 B 202
202 J 349
349 J 17
17 J 197
```

```
1. import random
2.
3. def irány(régi_irány, új_irány):
4.     if ((új_irány - régi_irány + 360) % 360 < 180):
5.         return 'J'
6.     elif ((új_irány - régi_irány + 360) % 360 > 180):
7.         return 'B'
8.     else:
9.         return random.choice(['B', 'J'])
10.
11. napló = []
12. with open('hajonaplo.txt') as forrásfájl:
13.     for sor in forrásfájl:
14.         sor = sor.strip().split('-')
15.         sor = list(map(int, sor))
```

Úgy szokás, hogy az importálást követően adjuk meg a függvényeinket, eljárásainkat.

az előző oldalon olvasható pszeudokód megvalósítása

darabolás a kötőjel mentén

```
16.         napló.append(sor)
17.
18. with open('kormanyzasok.txt', 'w') as célfájl:
19.     for index in range(len(napló)-1):
20.         print(napló[index][0],
21.             irány(napló[index][0], napló[index+1][0]),
22.             napló[index+1][0],
23.             file = célfájl)
```

Ezúttal mindenképp át kell alakítani a számokat tartalmazó karakterláncokat!

26. példa: Távirat

2021. április 29-én lehetett utoljára táviratot feladni a magyar postahivatalokban. Ekkor már régen nem morzekóddal továbbították a jeleket.

Volt idő, amikor a morzekódokkal csak az angol ábécé (nagy)betűit és a számjegyeket lehetett kódolni, így a magyar ékezetes betűk helyén több betűből álló összetételeket küldtek. Az *á*-ból *aa,* az *é*-ből *ee* lett, és volt még *ii, oo, uu.* Az *ö* helyett *oe,* az *ő* helyett *ooe,* az *ü* helyett *ue,* az *ű* helyett *uue* került a szövegbe. A mondatvégi írásjelek helyett a STOP karaktersor morzekódját küldték át az éteren vagy a kábelen, a mondat belsejében lévőkről pedig lemondtak.

A „Förgeteg közeledik, előz bennünket!” mondat távirati alakja, ahogy a postás a távíróval elküldte, és ahogy a címzett olvashatta, a következő volt: „FOERGETEG KOEZELEDIK ELUUEZ BENNUENKET STOP”. Az „Áá, dehogy!” pedig ezt a formát öltötte: „AAAA DEHOGY STOP”.

Írjunk programot `tavirat.py` néven, amely a felhasználótól bekért szöveget a fent jelzett formára alakítva írja vissza a képernyőre!

Programunkban nem minden karakternek feleltetünk meg valamit, azaz lényegében kiválogatjuk azokat, amelyeknek lesz megfelelőjük. Ezt egy másolás követi: az ékezetmentes karakterek és a szóközök helyett saját magukat másoljuk vissza, az ékezetesek helyett a nekik megfelelő összetételt, a mondatvégi írásjelek helyett pedig szóközt és a STOP karaktersort.

A feladat ugyanakkor felfogható egyetlen másolásnak is: a mondat belsejében lévő írásjelek helyére üres karakterláncot, üres karaktert másolunk.

Ha az általunk használt programozási nyelv kínál egyszerű módot szövegek nagybetűssé alakítására, akkor használjuk azt! Ha nem, akkor ezt egy újabb másolással kell megoldanunk. A nagybetűssé alakításnak meg kell előznie a távirati szöveggé való alakítást.

A program lényegi és jól elkülöníthető része az átírt szövegváltozat előállítása. Szervezzük ezt ki függvénybe, amelynek paramétere a nagybetűs karakterlánc, visszatérési értéke pedig az átalakított karakterlánc! Az átalakítási szabályokat csak a függvénynek kell ismernie, a megfeleltetéseket tároló adatszerkezetet helyezzük el a függvény belsejében!

A feladatot először úgy oldjuk meg, hogy nagyon kevés, a modern nyelvekre jellemző módszert használunk:

```
1. def táviratttá_alakít(szöveg):
2.     szövegközi_írásjelek = ',;:"'
3.     írásjeltelen = []
4.     for karakter in szöveg:
5.         if karakter not in szövegközi_írásjelek:
6.             írásjeltelen.append(karakter)
7.     írásjeltelen = ''.join(írásjeltelen)
```

KIVÁLOGATjuk azokat a karaktereket, amelyeket megtartunk.

```
 8.
 9.     helyettesítések = [['Á', 'AA'], ['É', 'EE'], ['Í', 'II'],
10.                        ['Ó', 'OO'], ['Ú', 'UU'], ['Ö', 'OE'],
11.                        ['Ő', 'OOE'], ['Ü', 'UE'], ['Ű', 'UUE'],
12.                        ['.', ' STOP'], ['?', ' STOP'], ['!', ' STOP']]
13.     távirat = []
14.     for karakter in írásjeltelen:
15.         volt_helyettesítés = False
16.         for helyettesítés in helyettesítések:
17.             if karakter == helyettesítés[0]:
18.                 távirat.append(helyettesítés[1])
19.                 volt_helyettesítés = True
20.                 break
21.         if not volt_helyettesítés:
22.             távirat.append(karakter)
23.     távirat = ''.join(távirat)
24.     return távirat
25.
26. átalakítandó = input('Mi lesz a távirat szövege? ')
27. print('Táviratként:', távirattá_alakít(átalakítandó.upper()))
```

Ha szükséges cserélnünk, akkor a helyettesített karaktert másoljuk a távirat végére...

...különben az eredetit.

A nagybetűs üzenet lesz a paraméter a függvény hívásakor.

A másik megoldásunkban törekedtünk a használt programozási nyelv speciális eszközkészletének mind tökéletesebb kihasználására. Míg az előző kódot Pythonban nem, de más nyelven programozni tudó ember is hamar el tudja olvasni, meg tudja érteni, át tudja írni az általa használt nyelvre, a most következővel lényegesen nehezebben boldogulna. Az újabb megoldás előnye, hogy a nyelvhez értők sokkal könnyebben olvassák, módosítják, mert lényegesen letisztultabb. Minden modern programozási nyelvben megtaláljuk a csak az adott nyelvre jellemző sajátos megoldásokat, utasításokat.

A kód nagy része ez a jegyzék.

```
 1. def távirattá_alakít(szöveg):
 2.     helyettesítések = {'Á': 'AA', 'É': 'EE', 'Í': 'II',
 3.                        'Ó': 'OO', 'Ú': 'UU',
 4.                        'Ö': 'OE', 'Ő': 'OOE',
 5.                        'Ü': 'UE', 'Ű': 'UUE',
 6.                        '.': ' STOP', '?': ' STOP', '!': ' STOP',
 7.                        ',': '', ';': '', ':': '', '"': ''}
 8.     for helyettesítendő, helyettesítő in helyettesítések.items():
 9.         szöveg = szöveg.replace(helyettesítendő, helyettesítő)
10.     return szöveg
11.
12. print('Íme:', távirattá_alakít(input('Mi lesz a szöveg? ').upper()))
```

Kiválogatás helyett másolunk.

A cseréket olyan ciklus végzi, amelynek magja egyetlen sor. A replace() tagfüggvény a háttérben a fenti kódunk ciklusmagjához hasonló műveletsort végez. Csak nem látjuk.

Mindent bele! – Összefüggő feladatsor megoldása az eddigi nyelvi készletünkkel

Ahogy egy szép, harmatos reggelen szertenézünk nagy tölgyek alatt megbúvó, fehérre meszelt tanyánkon, kérdések és feladatok özöne kezd bennünket nyugtalanítani:

1. Hány állatunk van összesen?
2. Melyik állatunk a legöregebb?
3. Van-e olyan fajú állatunk, amelyet a felénk járó postás (a felhasználó) kérdezett?
4. Írjuk ki az állatfajok listáját! Kérjünk be egy fajnevet a felhasználótól! Írjuk ki, hogy az egyéves állataink között van-e ilyen fajú!
5. Melyik állatfajhoz tartozó állatból van a legtöbb?
6. Mennyi az állataink átlagéletkora fajonként?

És még egy feladatunk van:

7. Írjuk ki az egyes állatok neveit a fajuknak megfelelő nevű szövegfájlba!

Rendelkezésünkre áll a már ismert `allatok.txt` fájl.
Írjuk meg a fenti feladatokat megoldó programot `mindent_bele.py` néven!
Az első dolgunk egészen biztosan a fájl beolvasása lesz. Az adatokat ezúttal állatonként egy-egy szótárban helyezzük el, a szótár kulcsai: `név`, `faj` és `kor`. Az első két kulcs értéke karakterlánc, az utolsóé egész szám. A szótárak egy `állatok` nevű listába kerülnek.

```
állatok = []
with open('allatok.txt') as forrásfájl:
    for sor in forrásfájl:
        sor = sor.strip().split()
        állat = {'név': sor[0], 'faj': sor[1], 'kor':int(sor[2])}
        állatok.append(állat)
```

az üres lista

típusátalakítás

A szótárat a lista végéhez fűzzük.

egy állatot tartalmazó szótár

Az **1. feladat** megoldása a szótár elemszámának meghatározása. A feladat megoldása egyetlen sor, mi mégis két sort közlünk. A második sor ugyanazt eredményezi, mint az első – azaz a feladatsor megoldásához bőven elég az egyik. Ebben a leckében azonban futólag megismerkedünk a Python legújabb módszerével a karakterláncok formázására, és f-string formában is megadjuk néhány feladat megoldását. Az f-stringek onnan ismerhetők fel, hogy *f*-fel kezdődnek. Talán legfontosabb tulajdonságuk, hogy a string *belsejében* is szerepelhetnek változónevek vagy kiszámolandó kifejezések, mégpedig {kapcsos zárójelben}.

```
print(len(állatok), 'állatunk van összesen.')
print(f'{len(állatok)} 'állatunk van összesen.')
```

hagyományos megoldás

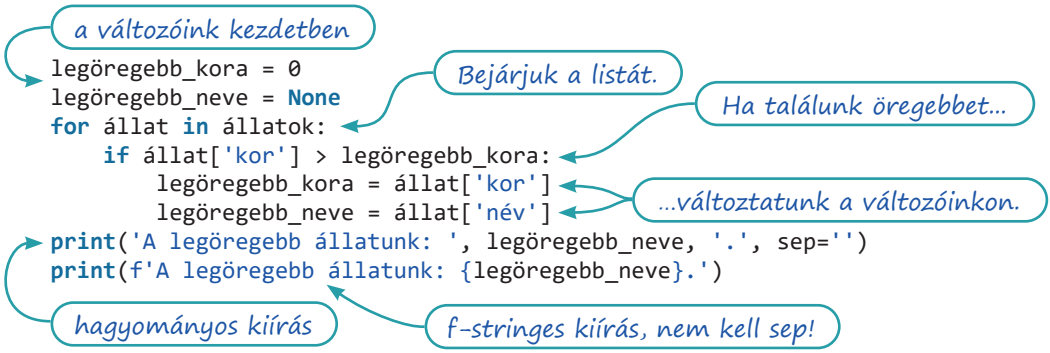
két rész, vesszővel felsorolva

egyetlen string

az f-string kezdő f-je

kiszámolandó kifejezés {}-ben

A **2. feladat** egy maximumkiválasztás.



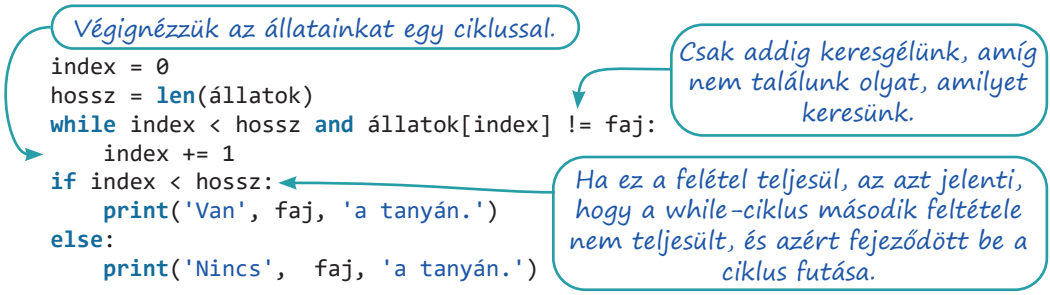
Minden magyarázat nélkül csak megjegyezzük, hogy a fenti sok sor helyett van jóval pythonosabb, a fentivel egyező eredményt adó megoldás is:

```
legöregebb_neve = max(állatok, key=lambda x: x['kor'])['név']
print(f'A legöregebb állatunk: {legöregebb_neve}.')
```

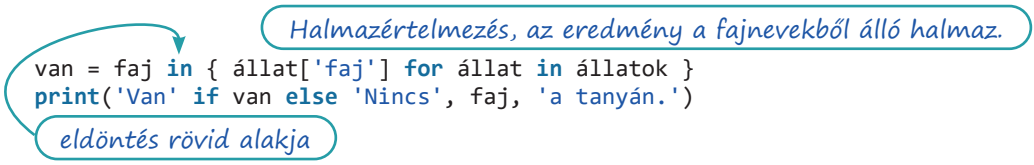
A **3. feladatban** bekérjük a fajt a postástól:

```
faj = input('Adjon meg egy fajnevet! ')
```

Innentől a feladat egy eldöntés.

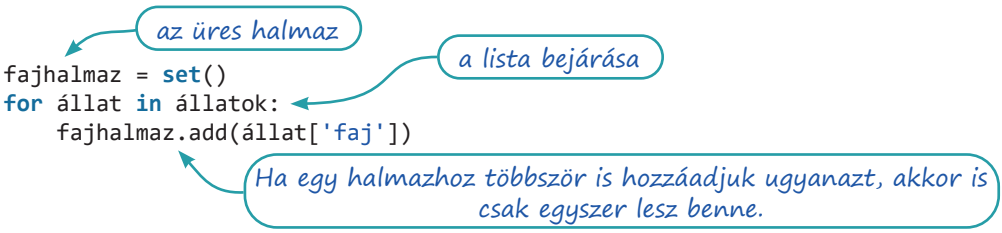


Egy, a fentivel azonos eredményt adó, igazán pythonos megoldás:



Természetesen a másodikként ismertetett megoldás kevésbé hatékony, mivel nem lépünk ki `break`-kel a halmaz előállításából, ha már találtunk a keresett fajú állatból. Ez a különbség csak nagyon-nagyon sok elemű adatszerkezetek átnézésekor lesz jelentős.

A **4. feladat** első részében már muszáj előállítanunk azt a halmazt, amelyet az imént csak ínyenceknek szóló megoldásmintánkban láttunk. Mindenképp halmazt érdemes előállítanunk, hiszen egy listában többször is előfordulna ugyanaz a faj. Ezúttal a hagyományos módon felírt ciklussal fogunk a feladathoz.




```
fajhalmaz = set()
for állat in állatok:
    fajhalmaz.add(állat['faj'])
```

Kiírjuk a halmaz elemeit (a tanyánkon előforduló fajokat), majd bekérjük a felhasználótól, hogy melyik érdekli.

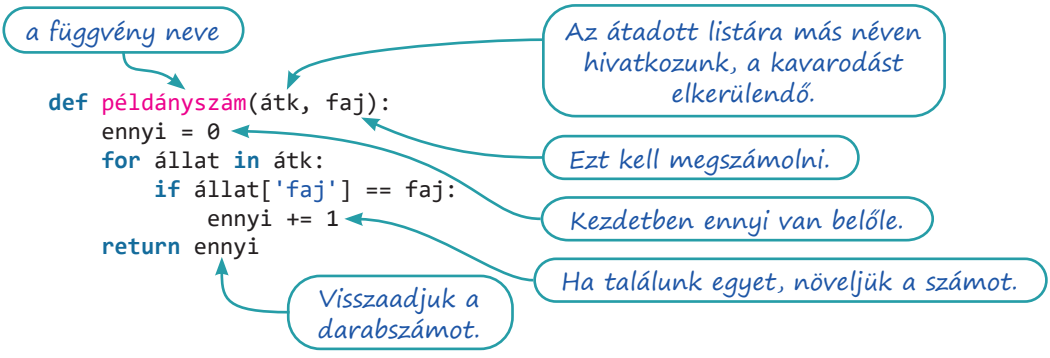

```
print('Állatfajok listája: ', ', '.join(fajhalmaz), '.', sep='')
print(f'Állatfajok listája: {", ".join(fajhalmaz)}.')
faj = input('Válasszon egy fajnevet! ')
```

Az f-string használatán alapuló megoldásban azért kell idézőjel az aposztróf helyett, mert az aposztróf lezárná magát a stringet, és ezt mindenképp el akarjuk kerülni.

Ezt követően már csak egy, az előző feladathoz hasonló eldöntés következik. A különbség az, hogy ezúttal két feltételünk is van: az állat faja és a kora.

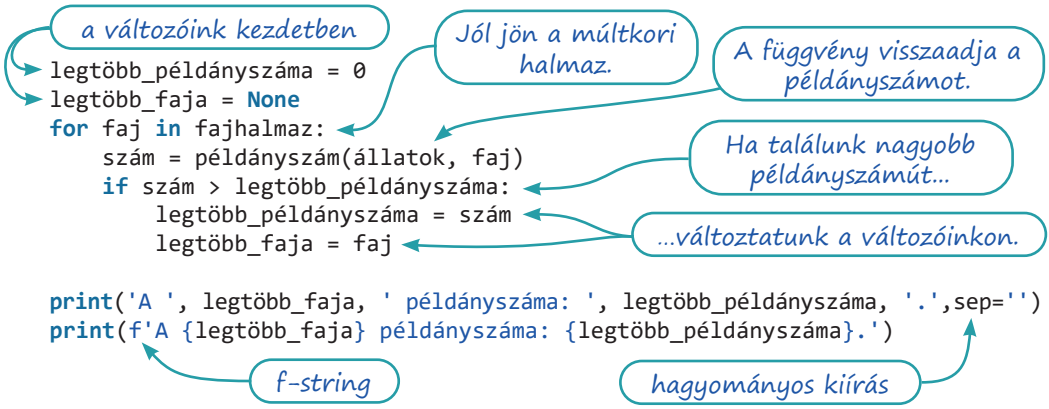
```
van = False
for állat in állatok:
    if állat['faj'] == faj and állat['kor'] == 1:
        van = True
print('Van' if van else 'Nincs', 'egyéves', faj, 'a tanyán.')
```

Az **5. feladatban** eljön az ideje egy függvény megírásának, hiszen egy olyan számlistának kell kiválasztanunk a maximumát, amelyet előbb létre kell hoznunk. A `példányszám` függvény megszámolja, hogy a megadott fajnak hány képviselője él a tanyán. Két paramétert vár: az `állatok` listát és egy fajnevet.



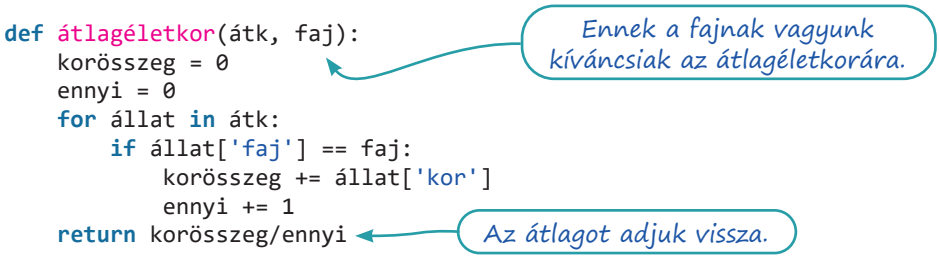

```
def példányszám(átk, faj):
    ennyi = 0
    for állat in átk:
        if állat['faj'] == faj:
            ennyi += 1
    return ennyi
```

A fenti függvényt a főprogramban hívjuk. Ez a programrész egy maximumkiválasztás, de a maximális érték megkeresésével párhuzamosan megkeressük a hozzá tartozó állatfajt is – ezzel megspórolunk egy külön keresést.



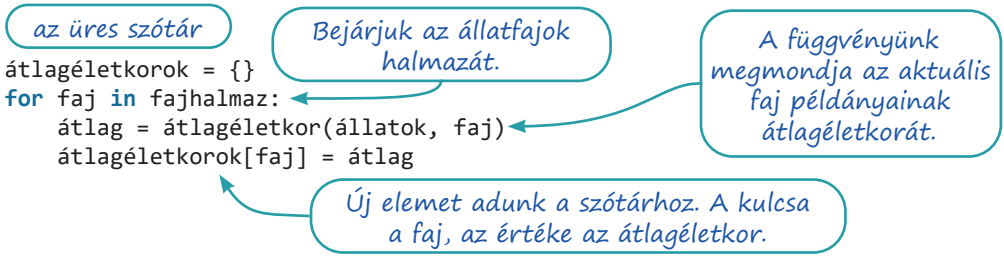

```
legtöbb_példányszáma = 0
legtöbb_faja = None
for faj in fajhalmaz:
    szám = példányszám(állatok, faj)
    if szám > legtöbb_példányszáma:
        legtöbb_példányszáma = szám
        legtöbb_faja = faj

print('A ', legtöbb_faja, ' példányszáma: ', legtöbb_példányszáma, '.',sep='')
print(f'A {legtöbb_faja} példányszáma: {legtöbb_példányszáma}.')
```

A **6. feladat** megoldását is egy függvény megadásával kezdjük. A függvény a paraméterként átadott faj átlagéletkorát adja vissza.




```
def átlagéletkor(átk, faj):
    korösszeg = 0
    ennyi = 0
    for állat in átk:
        if állat['faj'] == faj:
            korösszeg += állat['kor']
            ennyi += 1
    return korösszeg/ennyi
```

A fenti függvény hívásával egy szótárat állítunk elő a főprogramban. A szótár kulcsai az állatfajok, az értékek az átlagéletkorok.




```
átlagéletkorok = {}
for faj in fajhalmaz:
    átlag = átlagéletkor(állatok, faj)
    átlagéletkorok[faj] = átlag
```

Ezúttal külön-külön mutatjuk be a hagyományos kiírást és az f-string használatával működőt. Természetesen mindkettő az új szótárunk kulcs-érték párjainak bejárásán alapul.


```
for faj, átlagéletkor in átlagéletkorok.items():
    print(faj, ': ', átlagéletkor, sep='')
```

Az f-string lehetőséget nyújt a kiírt adatok balra, jobbra és középre rendezésére. A rendezések közül a <, a > és a ^ karakterrel adhatjuk meg a nekünk tetszőt. Megmondhatjuk azt is, hogy hány karakter széles lehet a rendezéssel kialakuló oszlop, és számok esetén a megjelenítendő tizedesjegyek számát is. Ezt használjuk most ki.

A fajnevet 14 karakteren, jobbra rendezve (figyeljük a csibecsőrt) írjuk ki.

```
for faj, átlagéletkor in átlagéletkorok.items():
    print(f'{faj:>14}: {átlagéletkor:>4.2f}')
```

Az átlagéletkort 4 karakteren, 2 tizedesre kerekítve, szintén jobbra rendezve írjuk ki.

Ehhez hasonló eredményt kell kapnunk:

```
        birka:  3.50
       malac:  1.67
        liba:  2.00
       kakas:  1.00
       kutya:  8.00
      kecske:  5.00
      macska:  2.50
  szarvasmarha:  3.00
       kacsa:  1.50
```

Kerekített érték – hasonlítsuk össze a régimódi kiírásnál látottal!

A **7. feladatban** két, egymásba ágyazott ciklusunk lesz. A külső végiglépked a fajok halmazának tagjain, a belső pedig az `állatok` listában keresi az épp aktuális fajhoz tartozó egyedeket.

Bejárjuk az állatfajok halmazát.

Megnyitjuk a fajnévnek megfelelő fájlt.

```
for faj in fajhalmaz:
    with open(faj + '.txt', 'w') as célfájl:
        for állat in állatok:
            if állat['faj'] == faj:
                print(állat['név'], file=célfájl)
```

Végigjárjuk az „állatok” listát.

Ha a fajba tartozó állatot találunk...

...a nevét beleírjuk a fájlba.

Megválaszoltuk a kérdéseket, felszáradt a harmat is, kezdődhet a nap!

Rend a lelkünk – A rendezés típusalgoritmusa és rendezés a napi gyakorlatban

Adatokat számítógéppel rendezni mind a mai napig az informatika egyik legérdekesebb problémája. Mindennapos élményünk, hogy egy weboldal – kereső, webshop, internetes adatbázis – sok ezer találatot, terméket, szócikket jelenít meg egyik-másik szempont szerint rendezve. A rendezéseknek ilyenkor nagyon gyorsan kell megtörténniük, különben a weboldal látogatója elunja a várakozást, és továbbáll. Az alkalmazások tervezői praktikák sokaságát vetik be, hogy a rendezett eredmények minél gyorsabban megjelenjenek, de a gyors eredmény alapja mindig a megfelelően megválasztott, hatékony rendezési típusalgoritmus.

Ha megijednénk, hogy ezek szerint rendezésből több is van, akkor megnyugtatásul elmondjuk, hogy bár az ijedelem megalapozott, találunk majd megnyugtató megoldást.

Egyszerű cserés rendezés

Sokféle rendezési típusalgoritmus létezik, amelyek nemcsak működésükben különböznek, hanem abban is, hogy milyen adathalmazt rendeznek gyorsan, hatékonyan. A hatékonyabb algoritmusok általában bonyolultabbak is, de mi most csak a legegyszerűbbel ismerkedünk meg. Számunkra ez már csak azért is elegendő, mert a mi adataink elemszáma elég kicsi, így nincs égető szükségünk hatékony rendezésre.

Az egyszerű cserés rendezés lényege, hogy az adatsor első elemét összehasonlítjuk az összes mögötte lévővel, és amelyik a későbbiek közül kisebb, azzal kicseréljük. Így a nagyobb elem az adatsor vége felé mozdul, a kisebb pedig az eleje felé. Ugyanezt a műveletet megismételjük az adatsor második, harmadik és az összes többi elemével, így a végén rendezett adatsort kapunk.

Futtassuk az alábbi programot, és nézzük meg a lista alakulását!

```
1. lista = [5, 3, 9, 1, 7]
2.
3. for egyik in range(len(lista)-1):
4.     for másik in range(egyik+1, len(lista)):
5.         print(lista, 'ezeket hasonlítom:', egyik, másik, end=' ')
6.         if lista[egyik] > lista [másik]:
7.             lista[egyik], lista[másik] = lista[másik], lista[egyik]
8.             print('csere volt, új sorrend:', lista[egyik], lista[másik],
9.                   'a lista a csere után:', lista)
10.        else:
11.            print('nem cserélünk')
```

Mindegyik elemet összehasonlítjuk...

...az összes mögötte lévővel.

Ha kell, cserélünk.

A belső ciklus mindig a *soron következő indexű* helyre keresi meg az oda való elemet. Amikor először fut végig, a 0. indexű helyre kerül a legkisebb elem. Amikor másodszor fut végig, az 1. indexű helyre kerül a második legkisebb elem. És így tovább.

Ezt és sok más algoritmust is *eltáncol* a Maros Művészegyüttes a Sapientia Erdélyi Magyar Tudományegyetem munkatársai rendezésében készült videókon, ezenkívül további érdekes animációkat is érdemes megnézni az interneten (például http://anim.ide.sk/rendezesi_algoritmusok_1.php).

A sorted() függvény és a lista.sort() tagfüggvény

Most, hogy van elképzelésünk a rendezés működéséről, lássuk a megnyugtató megoldást! A Pythonnak és a modern programozási nyelveknek általában van beépített parancsuk, függvényük a rendezésre. A Pythonnak rögtön kettő is. Nézzük meg a működésüket, futtassuk az alábbi programot!

```
1. lista = [5, 3, 9, 1, 7]
2.
3. rendezett = sorted(lista)
4. print(lista, rendezett)
5.
6. fordítva_rendezett = sorted(lista, reverse=True)
7. print(lista, fordítva_rendezett)
8.
9. lista.sort()
10. print(lista)
11.
12. lista.sort(reverse=True)
13. print(lista)
```

Az első paranccsal az eredeti lista nem változik, a rendezés eredményét nekünk kell kiírni vagy új listába tenni.

Fordított irányba is tudunk rendezni.

A második módszer az eredeti listában rendezi sorba az elemeket. Úgy mondjuk, hogy helyben rendez. Az eredeti sorrend elvész.

Ezzel is tudunk fordított irányba rendezni.

Természetesen a rendezés szövegek esetében is működik. A rendezés alapját alapesetben a karakterek kódolás szerinti sorrendje képezi – arról a kódolásról van szó, amelyről a fájlbeolvasáskor már szót ejtettünk. A lényeg – ha csak a latin ábécén alapuló nyelveket tekintjük – az, hogy az ékezetes betűt nem tartalmazó karakterláncok minden további nélkül rendezhetők. Az angol ábécé betűi ugyanis az ábécésorrendnek megfelelő sorrendben kaptak számkódokat. Az „A” kódja kisebb, mint a „B” kódja, és így tovább. A kisbetűk a nagybetűk után következnek, azaz a „Z” előbb következik, mint az „a”. Ha a rendezendő szavak eleje egyezik, a `sorted()` függvény ügyesen megnézi a többi karaktert is. Próbáljuk ki!

```
1. lista = ['Dalmata', 'bakancs', 'alma', 'alap']
2. print(sorted(lista))
```

Az egyes nyelvek ábécéjének figyelembevétele rendezéskor már egészen más szintű probléma. A megoldáshoz egyes programozási nyelvekben ügyeskednünk kell egy keveset, másokban pedig jó sokat – de ezt a problémakört meghagyjuk azoknak, akik a középiskola elvégzését követően is szívesen programoznának.

27. példa: A tanyán élő állataink neve és kora

Újabb példánkban tanyánk állatai közül a kutyák és a macskák szerepelnek, no meg a kakasok – rájuk mindig számíthatunk, ha nem akaródzik hajnalban kelni. Írjuk ki a nevüket a `rendezett_allatok.py` programban előbb ábécérendben, majd kor szerinti sorrendben, a legöregebbtől a legfiatalabbig! Utóbbi esetben kihasználhatjuk, hogy nincs két azonos korú kedvencünk.

Gondoljuk át, miként oldjuk meg ezt a feladatot!

- Beolvassuk a fájl adatait, és tároljuk őket egy kétdimenziós listában – ezt már többször megoldottuk, csak le kell másolnunk valamelyik régebbi programunk elejét.
- Első részfeladat:
  - Gyűjtsük listába a kutyák, a macskák és a kakasok nevét!
  - Rendezzük a listát!
  - Írjuk ki a rendezett névlista elemeit!
- Második részfeladat:
  - Gyűjtsük listába a három faj egyedeit (nemcsak a nevet, hanem az egész kis listát, mert a korra is szükség lesz)!
  - Gyűjtsük ki a korokat!
  - Rendezzük a korokat fordított sorrendben!
  - Járjuk be a korokat, keressük meg és írjuk ki az egyes korokhoz tartozó állatokat!

(Ha szívesebben dolgozunk listákat tartalmazó listákkal, akkor megtehetjük, hogy nemcsak az állatok nevét gyűjtjük ki az első lépésben, hanem az egyes állatok kis listáit is, és így oldjuk meg a feladatot.)

Kódoljuk a feladat megoldását!

```
 1. állatok = []
 2.
 3. with open('allatok.txt') as forrásfájl:
 4.     for sor in forrásfájl:
 5.         sor = sor.strip().split()
 6.         sor[-1] = int(sor[-1])
 7.         állatok.append(sor)
 8.
 9. nevek = []
10. for állat in állatok:
11.     if állat[1] in ['kutya', 'macska', 'kakas']:
12.         nevek.append(állat[0])
13.
14. print('Kedvenceink ábécérendben:\n', '\n'.join(sorted(nevek)), sep='')
15.
16. korok = []
17. kedvencek = []
18. for állat in állatok:
19.     if állat[1] in ['kutya', 'macska', 'kakas']:
20.         korok.append(állat[-1])
21.         kedvencek.append(állat)
22.
23. korok.sort(reverse=True)
24.
25. print('Kedvenceink kor szerint:')
26. for kor in korok:
27.     for kedvenc in kedvencek:
28.         if kor == kedvenc[-1]:
29.             print(kor, kedvenc[0])
30.
```

Tároljuk a fájlban lévő adatokat.

Kigyűjtjük a neveket.

rendezés

Az ábécésorrendben lévő neveket sortöréssel elválasztva írjuk ki.

Kigyűjtjük a korokat.

Kigyűjtjük a kedvenc állatainkat.

Rendezzük a korokat.

Megkeressük és kiírjuk az egyes korokhoz tartozó állatneveket.

Megjegyezzük, hogy a Python `sort()` és `sorted()` függvénye egyébiránt remek dolgokat tud, például a *kedvencek* kétdimenziós listát a kis listák utolsó eleme – a kor – alapján akár fordított irányba is képes rendezni. A szintaxis nem egyszerű, de az érdeklődők számára megmutatjuk:

```
kedvencek.sort(key=lambda x: x[-1], reverse=True)
```

Eggyé válás – A metszetképzés és az egyesítés (unió) típusalgoritmusa

Típusalgoritmusokkal foglalkozó leckéink közül az utolsóban két olyan módszerrel ismerkedünk meg, amelyek nem is egy, hanem két vagy több kiindulási adatszerkezethez rendelnek egyetlen újat.

A metszetképzés és az unió valójában egyaránt halmazművelet. Két halmaz metszete egy olyan halmaz, amelyik a két halmaz közös elemeit tartalmazza. Két halmaz uniója egy olyan halmaz, amelyik a két halmaz minden elemét tartalmazza – a közöseket is, meg azokat is, amelyek csak az egyik kiindulási halmazban vannak benne.

Vannak olyan programozási nyelvek, amelyekben nincs halmaz adattípus. Az ilyenekben a halmazokat valamilyen egyéb bejárható adatszerkezettel – például listával, tömbbel – valósítjuk meg. Metszetképzéskor a két lista közös elemeit írjuk újabb listába, egyesítéskor pedig az egyik lista elemeit bővítjük a második listában lévő, az elsőben eddig nem szereplő elemekkel. A listák és a tömbök azonban két fontos dologban különböznek a halmazoktól:

- a listákban és a tömbökben előfordulhat azonos elem többször is, a halmazokban nem;
- a listákban és a tömbökben kötött sorrendjük van az elemeknek, a halmazokban nincs.

Metszetképzés

28. példa: Tömegközlekedés Százszorszépvárosban

Százszorszépvárosban szeretnénk közösségi közlekedéssel eljutni a 92-es villamos végállomásáról, a Fapapucs sugárútról a 19-es busz Balambér tér nevű megállójába. A két járat megállói listaként állnak rendelkezésünkre. Melyik megállókban tudunk átszállni a villamosról a buszra?

A közös megállók nevére van szükségünk, azaz metszetet kell képeznünk.

A következő műveleteket kell kódolnunk az `atszallas.py` nevű programban:

- Felveszünk egy üres listát, például `átszállások` néven.
- Ciklussal bejárjuk a villamos valamennyi megállóját.
- Ha az aktuális villamosmegálló a buszjáratnak is megállója, és még nem szerepel az `átszállások` listában, akkor hozzáfűzzük.

Kódoljuk a programot!

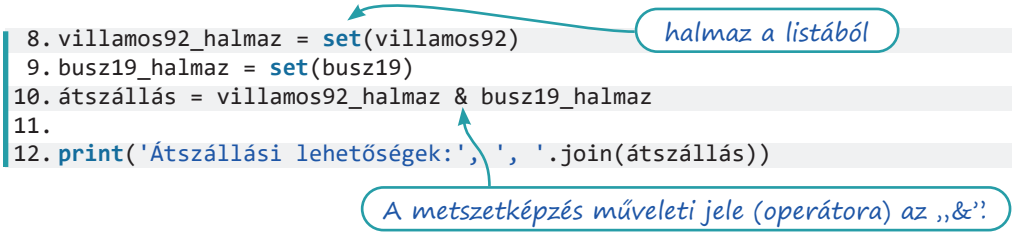
```
villamos92 = ['Kiss u.', 'Zöld fasor', 'Piros tér',
              'Malom-patak', 'Puccos u.',
              'Remegő-erdő', 'Fapapucs sgt.']
busz19 = ['Nagy u.', 'Balambér tér', 'Szent Lajos hídja',
          'Által-ér', 'Reghős Bendegúz sz.k.', 'Puccos u.',
          'Remegő-erdő', 'Pöszke-liget']

átszállás = []
for villamosmegálló in villamos92:
    for buszmegálló in busz19:
        if villamosmegálló == buszmegálló and \
           villamosmegálló not in átszállás:
           átszállás.append(villamosmegálló)
```

Az if villamosmegálló in busz19 kódrészlet felhasználásával egyszerűsíthetjük az algoritmust.

Vigyázzunk, hogy ne legyen ismétlődő elem!

```
14.             break
15.
16. print('Átszállási lehetőségek:', ', '.join(átszállás))
```

Ha a használt programozási nyelvnek van halmaz adattípusa, a dolgunk sokat egyszerűsödik. A kódot csak a 8. sortól közöljük, addig ugyanis nincs változás:




```
 8. villamos92_halmaz = set(villamos92)
 9. busz19_halmaz = set(busz19)
10. átszállás = villamos92_halmaz & busz19_halmaz
11.
12. print('Átszállási lehetőségek:', ', '.join(átszállás))
```

Említettük már, hogy a halmazokban nem definiált az elemek sorrendje. Futtassuk le a programunkat néhányszor, és látni fogjuk, hogy a két átszállási kapcsolatot nem minden alkalommal írja ki ugyanabban a sorrendben.

Említettük azt is, hogy a halmazokba nem kerülhet kétszer ugyanaz az elem. Ezt arra is kihasználhatjuk, hogy az ismétlődéseket tartalmazó listából eltüntessük az ismétlődéseket úgy, hogy a listát előbb halmazzá alakítjuk, majd a halmazt újra listává:

```
új_lista = list(set(ismétlődést_tartalmazó_lista))
```

A módszer nem használható, ha fontos az eredeti lista elemeinek sorrendje.

Unió

29. példa: Fricska és Kökény első szavainak jegyzéke

A szomszédban lakó két bölcsis – Nagy Fricska és Zsémbes Kökény – első hét, illetve nyolc szavát feljegyezték anyukáik. Közösen tartott névnapi zsúrjukon olyan dalt akarnak nekik énekelni, amely emléket állít első szavaiknak. Állítsuk össze azt a szójegyzéket, amelyben a két tipegő minden feljegyzett szava szerepel, ismétlődés nélkül!

A két jegyzék minden szavára szükségünk van, tehát az egyesítés típusalgoritmusát használjuk.

A következő műveleteket kell kódolnunk, ha nem használunk halmaz adattípust:

- Felveszünk egy üres listát.
- Az üres listába átmásoljuk az első lista elemeit.
- Az első lista elemeit tartalmazó listába átmásoljuk a második lista elemeit, figyelve, hogy csak azt másoljuk, ami még nincs benne.

Végezzük el a kódolást a `kozos_szavak.py` programban!

Mielőtt nekifogunk a kódolásnak, kitérünk a Python egyik, a listák másolásával kapcsolatos sajátságára. Ha csak annyit írunk a kódba, hogy

```
másolat = kiindulási_lista
```

akkor a másolat csak egy újabb hivatkozás lesz a számítógép memóriájában lévő eredeti listánkra. Ha ilyenkor a kiindulási listán változtatunk – mondjuk újabb elemet fűzünk

a végére, vagy kicserélünk egy már meglévő elemet –, akkor a változást a másolat is átveszi. A problémán úgy lehetünk úrrá például, hogy a fenti sor helyett a

```
másolat = kiindulási_lista[:]
```

parancsot adjuk ki.

```
 1. fricska = ['anya', 'maci', 'baba',
 2.            'apa', 'kaja', 'ló',
 3.            'erősáramú feszültségszabályzó']
 4. kökény = ['apa', 'autó', 'anya',
 5.           'víz', 'maci', 'könyv', 'nagyi',
 6.           'pörgettyűs tájoló']
 7.
 8. szójegyzék = fricska[:]
 9. for szó in kökény:
10.     if szó not in szójegyzék:
11.         szójegyzék.append(szó)
12.
13. print('Fricska első szavai:', ', '.join(sorted(fricska)))
14. print('Kökény első szavai:', ', '.join(sorted(kökény)))
15. print('A kész szójegyzék:', ', '.join(sorted(szójegyzék)))
```

Az első lista elemeit másoljuk az újba.

Új elemekként felvesszük Kökénynek azokat a szavait, amiket Fricska nem mondott.

Csak azért rendezünk, hogy könnyebben lássuk az eredményt.

Ha a használt programozási nyelv ismeri a halmaz adattípust, a dolgunk ezúttal is egyszerűsödik. A kódot csak a 8. sortól közöljük, addig megegyezik az előzővel.

```
 8. fricska_halmaz = set(fricska)
 9. kökény_halmaz = set(kökény)
10.
11. szójegyzék = fricska_halmaz | kökény_halmaz
12.
13. print('Fricska első szavai:', ', '.join(sorted(fricska_halmaz)))
14. print('Kökény első szavai:', ', '.join(sorted(kökény_halmaz)))
15. print('A kész szójegyzék:', ', '.join(sorted(szójegyzék)))
```

Az egyesítés műveleti jele (operátora) a „|” („cső”, Alt Gr+W magyar billentyűzetkiosztásnál).

Csoportnapló – Tapogatózás az objektumorientált programozás irányába

Ebben a leckében egyetlen program elkészítése a feladatunk. A feladatot az eddig tanult eszközeinkkel fogjuk megoldani, több alkalommal meg-megállva. Egy-egy ilyen megtorpanást arra használunk fel, hogy arról elmélkedjünk, milyen problémákkal szembesülünk, mennyire jó a megoldásunk, és hogy mi segíthetne jobbá tenni. Ha eltekintünk az elmélkedésektől, felfoghatjuk egyszerű gyakorlásnak is.

30. példa: Csoportnapló
Adott egy fájl az alábbi vagy hozzá hasonló tartalommal:

Mint azt látjuk, a fájl lehetne akár egy vegyes (több osztály diákjait is magába foglaló) tanulócsoport, például egy nyelvórai csoport tanárának feljegyzése a csoportba járó diákok jegyeiről.

A fájl beolvasását követően a következő feladatokat kell megoldanunk a `csoportnaplo.py` programban:

1. Írjuk ki a 11. zs osztályos diákok nevét!
2. Írjuk ki azoknak a nevét, akik nem írták meg mindhárom idei témazárót, és soroljuk fel a megírt témazáróiknak a jegyeit!
3. Határozzuk meg, ki írta a 11. x osztályban a legkevesebb témazárót!
4. Az utolsó előtti órán felelésre számítanak a diákok. A diákok szerint, amikor felelés van, az szokott felelni, akinek kevés a jegye. A tanárnő ennél pontosabb: azok közül válogat felelőt, akiknek a rendes jegyeinek száma kevesebb, mint a legtöbb rendes jeggyel bíró diák rendes jegyeinek 80 százaléka. Írjuk ki azoknak a nevét, akik „veszélyeztetettek”!
5. A tanárnő mégsem feleltet, hanem lezárja a diákok év végi jegyeit. Határozzuk meg és írjuk ki a diákok átlagát és év végi jegyét! Az átlagszámításkor a témazárók duplán számítanak. Az év végi jegy 1,7-es átlag fölött kettes, 2,5 fölött hármas, 3,5 fölött négyes, és 4,5-től ötös.

Gondoljuk végig, hogy az egyes kérdésekhez melyik típusalgoritmus szükséges!

1. kiválogatás
2. kiválogatások (hol van *t* betű a szám után) alapján megszámlálások (a témazárók száma); az eredmény alapján kiválogatás (nevek), végül újabb kiválogatások (a jegyek kiírása – elképzelhető, hogy az első kiválogatások eredményét használjuk majd)

3. minimumkiválasztás
4. megszámlálások, maximumkiválasztás, majd kiválogatás
5. sorozatszámítások

Mint ahogy már többször tudatosult bennünk, a jó adatszerkezet megkönnyíti a jó program írását. Határozzunk az adatok tárolásának módjáról!

Sok diák adatait kell tárolnunk. Egy-egy diáknak három *tulajdonságát* figyeljük meg:

- a nevét,
- az osztályát és
- a jegyeit.

A jegyek tűnnek a legmacerásabbnak, és elég sok művelet van velük kapcsolatosan. A forrásfájlban a témazárókra kapott jegyek egy listát alkotnak a rendes (nem témazáróra kapott) jegyekkel, alighanem azt a sorrendet tükrözve, ahogy a tanár feljegyezte őket. A kérdések között semmi nem vonatkozik a jegyek sorrendjére, az viszont, hogy témazáróra kapta-e a diák a jegyet, vagy sem, több esetben is fontos lesz. A jegyek elkülönítését az előzőek fényében érdemesnek tűnik már a fájl beolvasásakor megtenni. Érdemes-e már a beolvasáskor átalakítani a jegyeket számmá? Számolni fogunk velük, tehát érdemes.

Az osztályokkal kapcsolatos tevékenységeinket vizsgálva észrevesszük, hogy mindösszesze egyetlen feladatnál érdekes az évfolyam, azaz talán nem szükséges külön tárolnunk az évfolyamot és a betűjelet.

A neveket elég csak kiírni, és soha nem kell külön kezelnünk a vezeték- vagy keresztneveket. A nevek tárolhatók egyetlen karakterláncként.

Az eddigiek alapján érdemes lesz diákonként egy-egy szótárat létrehozni. Hangsúlyozzuk, hogy a feladat megoldható összetett listával is, de így tudunk az egyes *tulajdonságokra* név szerint hivatkozni.

- A szótár elsőként létrehozott kulcsa: „név”, az érték a diák neve.
- A második kulcs: „osztály”, az érték a diák osztálya olyan formában, ahogy a fájlban is szerepel.
- A harmadik kulcs: „rendes_jegyek”, az érték a rendes jegyeket egész számként tartalmazó lista.
- A negyedik kulcs: „témazáró_jegyek”, az érték az előzőhöz hasonló.

A beolvasás folyamán létrejött szótárakat egy listában helyezzük el.

A megoldás

Írjuk is meg a beolvasást végző részt! A közölt kód a 6. sortól kezdődik, mert elé még majd kerül egy függvény. Most az alábbi formában futtatható:

```
 6. diákok = []
 7. with open('naplo.txt') as forrásfájl:
 8.     for sor in forrásfájl:
 9.         sor = sor.strip().split(', ')
10.         diák = {}
11.         diák['név'] = sor[0]
12.         diák['osztály'] = sor[1]
13.         jegyek = sor[2].split()
14.         diák['rendes_jegyek'] = []
```

üres lista a szótáraknak

Soronként beolvassuk a fájlt.

A vessző + szóközök mentén darabolva háromtagú listává alakítjuk a sort.

A jegyeket tartalmazó részt szétdaraboljuk a szóközök mentén.

```
14.         diák['rendes_jegyek'] = []
15.         diák['témazáró_jegyek'] = []
16.         for jegy in jegyek:
17.             if len(jegy) == 1:
18.                 diák['rendes_jegyek'].append(int(jegy))
19.             else:
20.                 diák['témazáró_jegyek'].append(int(jegy[0]))
21.         diákok.append(diák)
```

Ha nincs t, akkor egy karakter hosszú egy jegy, ha van t, akkor kettő.

A t betűt nem tesszük el.

Az **1. feladatot** megoldó programrészlet (a sorok számozásakor számoljunk egy-egy, a kód olvashatóságát megkönnyítő üres sorral):

```
23. print('11. évfolyamra jár:')
24. for diák in diákok:
25.     if diák['osztály'][:2] == '11':
26.         print(diák['név'])
```

Itt állítjuk elő az évfolyamot az osztály jeléből.

A kód ennyi programozásóra után aligha ígért jelentősebb fejtörést, de alkalmat ad az első töprengésünkre.

Megegyeztünk abban, hogy a mostani programunk nem teszi szükségessé az osztály, azaz az évfolyam és a betűjel együttesének bonyolultabb tárolását. Nagyon könnyen el tudunk azonban képzelni olyan programot – elég csak az elektronikus naplóra gondolnunk –, ahol sok-sok programrész foglalkozik ezzel az adattal. Hol csak az évfolyamra van szükség, hol csak a betűjelre, hol mind a kettőre, de néha `11x`, máskor pedig `11. x` vagy `11/x` formában.

Természetesen az említett sok-sok programrész elő tudja magának állítani a megfelelő formát, pont úgy, ahogy a fenti kódrészlet 25. sorában mi is megtettük. De ez sok-sok újabb kódsort jelent, ami pedig – mostanra saját tapasztalataink alapján tudjuk – sok-sok hibalehetőséget hordoz magában. Milyen remek volna, ha lenne olyan, hogy `diák['évfolyam']` és `diák['oszálybetű']`!

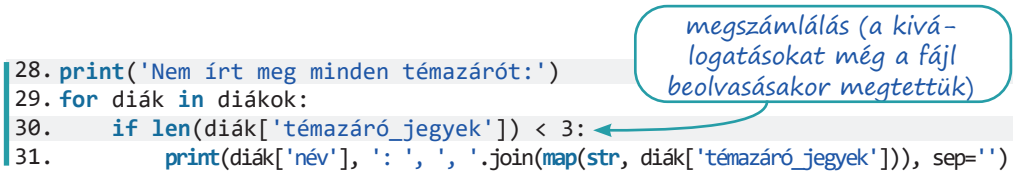
Ó, hiszen ilyet tudunk létrehozni! Egyszerűen újabb sorokat írunk a fájlbeolvasó részbe, és lesznek ilyen adataink!

Igen, létre tudunk hozni ilyen kódrészletet, de ne feledjük, hogy ilyenkor ugyanazt az adatot tároljuk többször. Ez legalább két okból problémás. Az egyik, hogy sok tárterületet foglal. A másik, hogy sok hibalehetőséget rejt magában. Ha például itt az év vége, és a `11x`-et `12x`-re kell javítani, akkor hány helyen is kell frissítenünk az adatokat? Előbb-utóbb valamelyikről elfelejtkezünk. Ez a megoldás tehát nem igazán jó megoldás.

Akkor írjunk rá függvényt! Lenne `évfolyam(diák)` függvényünk, átadnánk neki egy diákot (a diákot tartalmazó szótárt), a visszatérési érték pedig az évfolyam száma lenne.

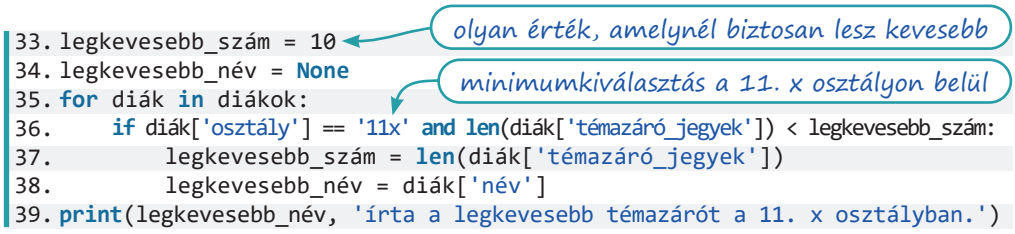
Ez a megoldás kiküszöböli az előző hiányosságait, cserébe viszont újakat teremt. Képzeljük el, hogy egy nagy programot fejlesztő programozói csoportba új ember érkezik. A kód hatalmas. Honnan fog rájönni az új programozó, hogy a diákok évfolyamát könnyen megkapja az `évfolyam()` függvény használatával? Honnan jövünk rá mi magunk, ha fél év után újra ezzel a kóddal kell foglalkoznunk? Még nem válaszolunk erre a kérdésre.

A **2. feladatot** a következő programrészlettel oldhatjuk meg:



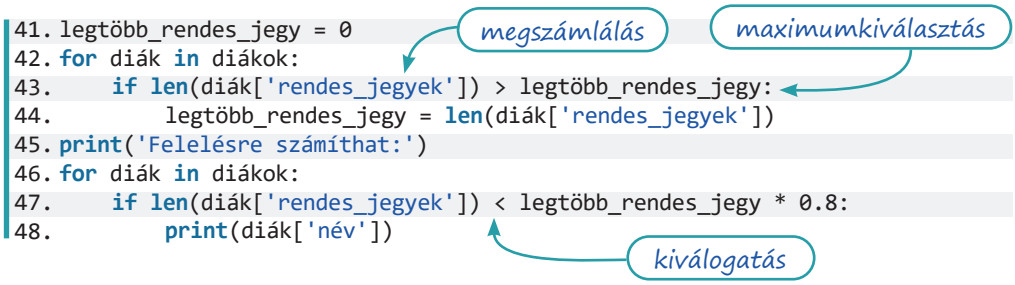

```
print('Nem írt meg minden témazárót:')
for diák in diákok:
    if len(diák['témazáró_jegyek']) < 3:
        print(diák['név'], ': ', ', '.join(map(str, diák['témazáró_jegyek'])), sep='')
```

A **3. feladatot** megoldó programrészlet:




```
legkevesebb_szám = 10
legkevesebb_név = None
for diák in diákok:
    if diák['osztály'] == '11x' and len(diák['témazáró_jegyek']) < legkevesebb_szám:
        legkevesebb_szám = len(diák['témazáró_jegyek'])
        legkevesebb_név = diák['név']
print(legkevesebb_név, 'írta a legkevesebb témazárót a 11. x osztályban.')
```

A **4. feladatot** a következő programrészlettel oldhatjuk meg:




```
legtöbb_rendes_jegy = 0
for diák in diákok:
    if len(diák['rendes_jegyek']) > legtöbb_rendes_jegy:
        legtöbb_rendes_jegy = len(diák['rendes_jegyek'])
print('Felelésre számíthat:')
for diák in diákok:
    if len(diák['rendes_jegyek']) < legtöbb_rendes_jegy * 0.8:
        print(diák['név'])
```

Ezen a ponton ismét megállunk töprengeni. Felidézzük azt a gondolatot, miszerint azért is fontos a könnyen olvasható program írása, mert egy kész programot manapság többször, sokszor olvas újra valaki azért, hogy megértse, mit csinál a program. Ez lehet az az érettségiző, aki reggel nyolckor programozással kezdte az érettségit, majd más feladatra váltott, és tizenegykor innen folytatná. De lehet az a fejlesztő is egy valós munkahelyen, akinek módosítania kell egy már működő és sokak által használt kódot, olyat, amit nem is ő írt.

Ha most ránézünk erre a kódra, látjuk-e szinte azonnal, hogy mit csinál? A 45. sor sokat segít, de bizonyosan további segítséget jelentene, ha a 47. sorban írhatnánk olyat, hogy `if diák.rendes_jegyek_száma()`. Szerencsére lesz erre lehetőség.

Az **5. feladat** megoldásához bontsuk a feladatot gondolatban két részre! Egyrészt meg kell határoznunk egy-egy diák átlagát, másrészt az átlag alapján megállapítani az év végi jegyét. Az átlag megállapítására függvényt írunk. Tudjuk, hogy a témazárók duplán számítanak, és szerencsére már a fájl beolvasásakor elkülönítettük a témazáróra kapott jegyeket a rendes jegyektől, most ezt kihasználva, külön listaként adjuk át a két jegylistát a függvényünknek. A függvény a következő formát ölti:

```
1. def átlag(rendes_jegyek, témazáró_jegyek):
2.     összeg = sum(rendes_jegyek) + sum(témazáró_jegyek) * 2
3.     osztó = len(rendes_jegyek) + len(témazáró_jegyek) * 2
4.     return round(összeg/osztó, 2)
```

Ahogy a sorszámozásból látszik, a kód elejére helyezzük el a függvényt – ami Pythonban nem kötelező, de általában nem bánjuk meg, ha így járunk el.

A kód végén a következőképpen hívjuk a függvényt:

```
50. for diák in diákok:
51.     á = átlag(diák['rendes_jegyek'], diák['témazáró_jegyek'])
52.     if á <= 1.7:
53.         jegy = 1
54.     elif 1.7 < á < 4.5:
55.         jegy = int(round(á, 0))
56.     else:
57.         jegy = 5
58.     print(diák['név'], 'átlaga:', á, 'jegye:', jegy)
```

Ezzel elkészültünk a feladat teljes megoldásával.

Utolsó töprengésünkben arra mutatunk rá, hogy ha nem mi írtuk a függvényt, akkor nem nyilvánvaló első pillantásra, hogy miért két paraméterrel kell hívnunk, és hogy mi ez a két paraméter. Mennyivel egyszerűbb volna azt írni a programunk 51. sorában, hogy `á = diák.átlag()`!

Gondoljuk egy kicsit tovább a problémát! Egy nagy programban – mint az említett e-napló – valószínűleg sok-sok helyen kell átlagot számolni. Tegyük fel, hogy egy napon az iskola elhatározza, hogy az órai munkára kapott jegyek az év végi átlag számításakor csak feleannyit érnek, mint egy „rendes” jegy. A fájl beolvasásakor hamar megoldható, hogy létrejöjjön egy `diák['órai_munka_jegyek']` lista, de ezt követően még *minden függvényhívást* módosítani kell, hiszen az `átlag()` függvénynek egy harmadik paramétert is át kell adnunk. Jó eséllyel valamelyiket elfelejtjük a sok közül, és akkor az e-napló egyik része szerint más lesz a diák átlaga, mint a másik rész szerint.

A töprengéseink során arra a megállapításra jutottunk, hogy

- elsősorban nagyobb programoknál,
- másodsorban a kód olvashatósága, azaz a kód átláthatósága és karbantarthatósága,
- végül a kód könnyebb bővíthetősége és a programozási folyamat egyszerűsödése miatt örülnénk, ha valamilyen, a fentieket megoldó eszközt kapnánk a kezünkbe. Szerencsére van ilyen eszköz, az úgynevezett objektumorientált programozás (röviden OOP), amely nem csak a fenti problémákra kínál megoldást.

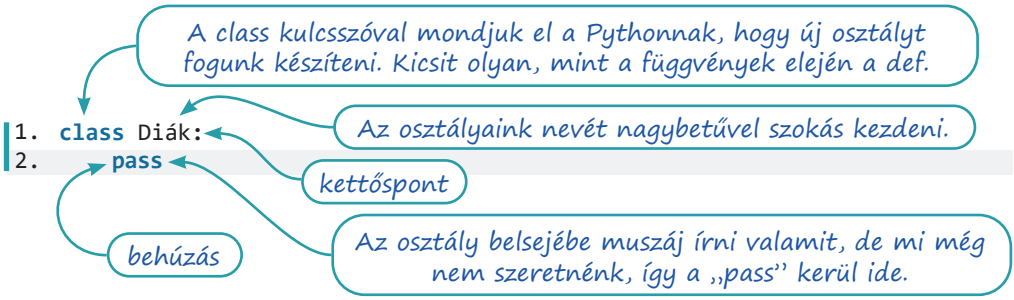
A módszer lényegének, fontosabb tulajdonságainak ismerete akkor is segítségünkre lehet, ha olyan rövid programokat írunk egymagunkban, mint amilyeneket eddig írtunk. Ennek az a magyarázata, hogy programjainkban gyakran támaszkodunk mások által megírt programrészekre. Tettünk már ilyet, a random modul használatával, de néhány leckével később grafikus felhasználói felülettel bíró programokat fejlesztünk majd, és ott nagyon hasznos, ha minél pontosabb elképzeléseink vannak az objektumorientált programok mibenlétéről.

A következő leckékben tehát az objektumorientált programozásról lesz szó.

A terv, amely nem megy füstbe – Saját adatszerkezetek tervezése és megvalósítása objektumosztályokkal

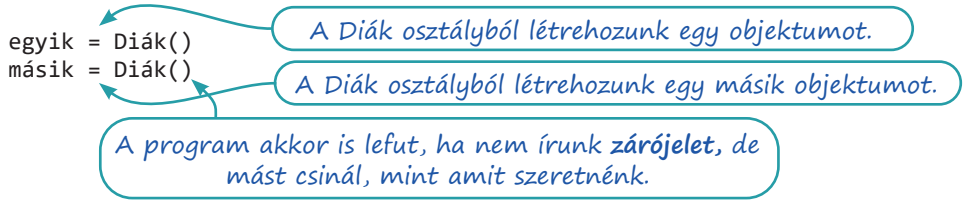
Eddig alapvetően háromféle összetett adattípussal – adatszerkezettel – dolgoztunk: listával, szótárral és halmazzal. Persze volt olyan listánk, amely listákat tartalmazott, és olyan is, amely szótárakat, illetve volt olyan szótárunk is, amelynek lista volt valamelyik értéke. Ebben a leckében megtanulunk a programunkhoz tökéletesen illeszkedő adatszerkezeteket létrehozni.

Egy ilyen, tökéletesen testreszabott adatszerkezet tervének neve: osztály, angolul: class. Lássunk egy ilyen tervet:



Az osztályokat szokás **objektumosztálynak** is nevezni, mert objektumokat hozunk belőlük létre. Az egy osztály alapján létrehozott objektumok elég hasonlóak, ezért azt mondjuk, hogy egy osztályba tartoznak.

Egészítsük ki a fenti kódot az alábbi két sorral! (A következő néhány példa sorait nem mindig számozzuk, mert csak próbálkozunk, gyakran fogunk törölni, új sorokat beírni.)



Van tehát két, `Diák` osztályú objektumunk. (Az, hogy ezek a diákok is iskolai osztályba járnak, puszta véletlen – ez egy másféle osztály.) Egy-egy objektumról azt mondjuk, hogy a tervéül szolgáló osztály egy példánya. Amikor létrehozzuk, akkor azt mondjuk, hogy **példányosítjuk** az osztályt.

A fenti példában a `Diák` tehát egy osztály, hosszabb nevén objektumosztály, azaz egy terv. Ennek a tervnek az alapján jön létre az `egyik` és a `másik` objektum, **objektumpéldány** vagy csak példány. Az utolsó két kódsor pedig két **példányosítás.** Az elsőben az `egyik`, a másodikban a `másik` nevű objektumot példányosítjuk a `Diák` osztályból.

De mi az az objektum?

Az objektumok egy-egy létező, azaz egy-egy személy, tárgy, fogalom számítógépes programban létrehozott reprezentációi, megvalósulásai. A létező lehet bármi: diák (mint a példánkban), kutya, egér, borsószem, érdemjegy, könyv, csavaranya, zeneszám, álom vagy éppen papucs orrán pamutbojt.

Az objektumokat ebben a leckében még csak arra használjuk, hogy a tárolt létezők bennünket érdeklő tulajdonságait tároljuk bennük. A tulajdonságok sokfélék lehetnek. A kutyának tárolhatjuk a nevét és a fajtáját, a borsószemnek a színét és az alakját, a könyvnek a szerzőjét, a címét és az oldalszámát, a zeneszámnak az előadóját és a hosszát. Az ilyen tulajdonságokat ebben a könyvben *jellemzőnek* nevezzük (szokás még többek között a mező, az adattag és a változó elnevezést használni).

Mi a diákoknak a vezeték- és a keresztnevét, valamint az évfolyamát tároljuk. Bővítsük a programunkat:

```
egyik.vezetéknév = 'Lopakodó'
egyik.keresztnév = 'Kasztanyetta'
egyik.évfolyam = 11
másik.vezetéknév = 'Surranó'
másik.keresztnév = 'Szalicil'
másik.évfolyam = 11
```

A tárolt jellemzőket ki is tudjuk írni:

```
print(egyik.keresztnév, másik.évfolyam)
```

A jellemzők értéke módosítható:

```
másik.évfolyam += 1
egyik.keresztnév = egyik.keresztnév + ' ' + 'Harmónia'
```

Az objektum inicializálása

Az objektumokat eddig úgy hoztuk létre, hogy példányosításukat követően az egyes jellemzőknek „kézzel” adtunk értéket. Ez a valóságban igen ritkán történik így. Sokkal gyakoribb, hogy az objektum jellemzőinek az objektum létrehozásakor adunk értéket.

Az értékadásra egy speciális függvény, az `__init()__` szolgál. Az `__init__()` függvény a példányosításkor automatikusan lefut, és végigcsinálja mindazt, amit beleírtunk. Úgy mondjuk, hogy ilyenkor történik meg az objektum inicializálása (magyarra előkészítésnek fordíthatnánk, de nem szokás fordítani). Néhány más programozási nyelvben az osztályok, objektumok hasonló szerepű függvényét konstruktornak nevezik, és ez az elnevezés olykor átszivárog a Python világába is.

Az osztály definíciója a következőképp módosul:

A függvény az osztály belsejében van.

```
1. class Diák:
2.     def __init__(self, vn, kn, évf):
3.         self.vezetéknév = vn
4.         self.keresztnév = kn
5.         self.évfolyam = évf
```

Az __init__() függvény első paramétere hagyományosan a self, és azt mondjuk el vele, hogy magán a függvényt tartalmazó objektumon dolgozzon a függvény.

Ezek az utasítások a függvény belsejében vannak.

A self miatt ezek a változók vagy jellemzők az objektum saját jellemzői lesznek. (Létrehozhatnánk olyan jellemzőket is, amelyek nem az egyes objektumokhoz, hanem az osztályhoz tartoznak, azaz minden, az osztályba tartozó objektum közös jellemzői.)

Az egyes objektumok példányosítása megváltozik:

Ezeket a paramétereket az __init__() függvény kapja meg. A „self”-et nem kell ideírnunk.

```
egyik = Diák('Lopakodó', 'Kasztanyetta', 11)
másik = Diák('Surranó', 'Szalicil', 11)
```

A többi utasítást pedig pontosan úgy használhatjuk, mint eddig. Próbáljuk is ki őket! Természetesen létre tudunk hozni új objektumot akár billentyűzetről, akár fájlból bekért adatok alapján:

```
v = input('Mi a harmadik diák vezetékneve? ')
k = input('Mi a harmadik diák keresztneve? ')
é = int(input('Mi a harmadik diák évfolyama? '))
harmadik = Diák(v, k, é)
```

Az általunk létrehozott objektumok is elhelyezhetők szótárban vagy – mint a példa mutatja – listában:

```
diákok = [egyik, másik, harmadik]
print(diákok[-1].keresztnév)
```

Feladatok

1. Írjunk programot `allataink.py` néven! Írjuk meg az `Állat` osztályt és az `__init__()` függvényét! A függvény a paraméterei alapján állítsa be jellemzőként az egyes példányok nevét, faját és korát!
2. Állítsunk elő `Állat` osztályú objektumokat az `allatok.txt` fájl alapján! Az objektumokat tároljuk az állatok nevű listában!
3. Járjuk be a listát, és írjuk ki a malacok nevét!

```
1. class Állat:
2.     def __init__(self, név, faj, kor):
3.         self.név = név
4.         self.faj = faj
5.         self.kor = kor
6.
7. állatok = []
8. with open('allatok.txt') as forrásfájl:
9.     for sor in forrásfájl:
10.        sor = sor.strip().split()
11.        állat = Állat(sor[0], sor[1], int(sor[2]))
12.        állatok.append(állat)
13.
14. for állat in állatok:
15.     if állat.faj == 'malac':
16.         print(állat.név)
```

Az állat és az Állat két különböző dolog a kis- és a nagybetű miatt. Az állat egy Állat osztályú objektum.

Ha elég szemfülesek vagyunk, észrevehettük, hogy a programot egyszerűbben, osztályok használata nélkül is megírhattuk volna (sőt, már írtunk is hasonlót pár leckével korábban):

```
állatok = []
with open('allatok.txt') as forrásfájl:
    for sor in forrásfájl:
        sor = sor.strip().split()
        állat = { 'név': sor[0], 'faj': sor[1], 'kor': int(sor[2]) }
        állatok.append(állat)

for állat in állatok:
    if állat['faj'] == 'malac':
        print(állat['név'])
```

Szótárakat hozunk létre, nem objektumokat.

A kiírás picit változik.

A Python szótár adattípusa remekül használható objektumok tárolására. Egy-egy objektumnak egy-egy szótár felel meg, a jellemzők nevei a szótár kulcsai, a jellemzők értékei a szótár kulcsaihoz tartozó értékek.

Akkor hát mi értelme osztályokkal meg objektumokkal vesződni? Két lecke múlva kiderül.

Az objektumokat tároló csoportnapló

Ebben a leckében az a feladatunk, hogy írjuk át a csoportnaplós programunkat, mégpedig úgy, hogy ne szótárakban, hanem saját magunk által írt osztályba tartozó objektumokban tárolja az egyes diákok adatait.

Tanulmányozzuk egy kicsit a programunk első kész változatát. Látható, hogy a fájl beolvasása, illetve egy-egy sor megfelelő darabolása elég jelentős része a programunknak. Megtehetjük-e, hogy az egyes sorokat már az osztályunk `__init__()` függvényével értelmeztetjük? Hát persze!

```
1. def átlag(rendes_jegyek, témazáró_jegyek):
2.     összeg = sum(rendes_jegyek) + sum(témazáró_jegyek) * 2
3.     osztó = len(rendes_jegyek) + len(témazáró_jegyek) * 2
4.     return round(összeg/osztó, 2)
5.
6. class Diák():
7.     def __init__(self, sor):
8.         self.név = sor[0]
9.         self.osztály = sor[1]
10.        jegyek = sor[2].split()
11.        self.rendes_jegyek = []
12.        self.témazáró_jegyek = []
13.        for jegy in jegyek:
14.            if len(jegy) == 1:
15.                self.rendes_jegyek.append(int(jegy))
16.            else:
17.                self.témazáró_jegyek.append(int(jegy[0]))
18.
19. diákok = []
20. with open('naplo.txt') as forrásfájl:
21.     for sor in forrásfájl:
22.         sor = sor.strip().split(', ')
23.         diák = Diák(sor)
24.         diákok.append(diák)
```

A függvény nem változik.

Ez a ,,sor'' már egy háromtagú lista.

A ,,jegyek'' és a ,,jegy'' változó ideiglenes, csak a függvényen belül létezik. Ezért nem kell eléjük self.

Amelyik változónak meg kell maradnia az __init__() lefutását követően is, annak self kerül a neve elé. Objektumváltozók, azaz jellemzők lesznek belőlük.

A fájlból beolvasott sort vessző + szóköz mentén daraboljuk.

Objektumot példányosítunk.

A kész objektumot a lista végére illesztjük.

A darabok listáját megkapja az osztály __init__() függvénye, amely a listában lévő adatok alapján feltölti az objektum jellemzőt.

```
25.
26. print('11. évfolyamra jár:')
27. for diák in diákok:
28.     if diák.osztály[:2] == '11':
29.         print(diák.név)
30.
31. print('Nem írt meg minden témazárót:')
32. for diák in diákok:
33.     if len(diák.témazáró_jegyek) < 3:
34.         print(diák.név, ': ', ', '.join(map(str, diák.témazáró_jegyek)), sep='')
35.
36. legkevesebb_szám = 10
37. legkevesebb_név = None
38. for diák in diákok:
39.     if diák.osztály == '11x' and len(diák.témazáró_jegyek) < legkevesebb_szám:
40.         legkevesebb_szám = len(diák.témazáró_jegyek)
41.         legkevesebb_név = diák.név
42. print(legkevesebb_név, 'írta a legkeveseeb témazárót a 11. x osztályban.')
43.
44. legtöbb_rendes_jegy = 0
45. for diák in diákok:
46.     if len(diák.rendes_jegyek) > legtöbb_rendes_jegy:
47.         legtöbb_rendes_jegy = len(diák.rendes_jegyek)
48. print('Felelésre számíthat:')
49. for diák in diákok:
50.         if len(diák.rendes_jegyek) < legtöbb_rendes_jegy * 0.8:
51.             print(diák.név)
52.
53. for diák in diákok:
54.     á = átlag(diák.rendes_jegyek, diák.témazáró_jegyek)
55.     if á <= 1.7:
56.         jegy = 1
57.     elif 1.7 < á < 4.5:
58.         jegy = int(round(á, 0))
59.     else:
60.         jegy = 5
61.     print(diák.név, 'átlaga:', á, 'jegye:', jegy)
```

Nem a szótár kulcsára, hanem az objektum jellemzőjére hivatkozunk.

Nem a szótár kulcsára, hanem az objektum jellemzőjére hivatkozunk.

Objektumaink működni kezdenek – Függvények az objektumok belsejében

Két leckével ezelőtt láttuk, hogy amikor csak adatszerkezetként használjuk az objektumokat, akkor nem igazán lépünk előre ahhoz képest, amikor egy-egy „létező” tulajdonságait egy-egy szótárban tároljuk. Python nyelven programozva így nem is tudtuk jelentőségének megfelelően értékelni ezt a lehetőséget. Sok más nyelvben nincs szótár vagy ahhoz hasonló képességű beépített adattípus, és már az adatszerkezetet megvalósító objektumot sem éreztük volna elhanyagolható előrelépésnek.

Az objektumok ereje ott mutatkozik meg igazán, amikor az osztályon, a terven belül függvényeket is írunk. Itt az ideje, hogy definíciót adjunk: **az objektumosztály egy adatszerkezet és az adatszerkezeten végezhető műveletek együttese.**

A „műveletek”-et elég tágan kell értenünk, mint az hamarosan kiderül.

31. példa: A macskák fejlődésének nyomon követése

Egy `macsek.py` nevű programfájlban hozzunk létre egy `Macska` nevű osztályt! A belőle példányosított objektumok az objektum létrehozásakor paraméterként kapják meg a macska nevét, születési évét és grammban kifejezett tömegét. Az `__init__()` függvény ezenfelül hozza létre a `tapasztalat` nevű jellemzőt, és adjon számára értékül egy 10 és 50 közötti véletlen számot! Ezzel a jellemzővel követjük majd nyomon a macskánk fejlődését. Minden sikeres vadászat növeli a tapasztalatot és ezzel a következő vadászat sikerének esélyét.

```
1. import random
2.
3. class Macska:
4.     def __init__(self, név, év, tömeg):
5.         self.név = név
6.         self.születési_év = év
7.         self.tömeg = tömeg
8.         self.tapasztalat = random.randint(10, 50)
```

Az osztályból objektumokat például a

```
cs = Macska('Csilus', 2018, 3652)
z = Macska('Zokni', 2017, 4003)
```

parancsok példányosítanak.

Az osztály műveleteit függvényekkel (eljárásokkal) valósítjuk meg. Már láttunk is egy ilyen függvényt, nevezetesen az `__init__()` nevűt. Az osztályok, objektumok belsejében létező függvényeket ebben a könyvben szinte mindig **tagfüggvénynek** hívjuk – már találkozhattunk a szóval. Ez a név kifejezi, hogy ezek is függvények. Talán a legelterjedtebb elnevezésük a metódus, de szokás őket függvénytagnak is hívni.

Az első tagfüggvényünk arra való, hogy a macskáink „nyávogni” tudjanak. A tagfüggvény egyetlen paramétere az objektumra visszautaló `self`. Ahogy az `__init__()` esetén már láttuk, híváskor ezt a paramétert nem kell átadnunk.

```
 9.
10.     def nyávog(self):
11.         print('Nyaú!')
```

A tagfüggvényt az alábbi módon hívjuk:

```
cs.nyávog()
z.nyávog()
```

A második tagfüggvényünk azt szemlélteti, hogy a tagfüggvényeknek is lehet visszatérési értékük. A példában szereplő tagfüggvény egy jellemzőt is felhasznál a visszatérési érték kiszámításához.

```
12.
13.     def kor(self, mostani_év):
14.         return mostani_év - self.születési_év
```

A tagfüggvény használata 2022-ben:

```
print(cs.kor(2022))
```

A harmadik tagfüggvény arra mutat példát, hogy miként lehetséges egy jellemző értékének megváltoztatása tagfüggvény hívásával:

```
15.
16.     def eszik(self):
17.         self.tömeg += 25
```

A tagfüggvény hívásakor nem kell megadnunk paramétert:

```
print(cs.tömeg)
cs.eszik()
print(cs.tömeg)
```

Természetesen egy tagfüggvény többször is hívható:

```
print(z.tömeg)
for _ in range(10):
    z.eszik()
print(z.tömeg)
```

Az osztály utolsó előtti tagfüggvénye pedig azt példázza, hogy egy tagfüggvény egészen bonyolult műveleteket is megvalósíthat.

Felbukkan a véletlen tapasztalatú egér…

Programunk eposzi seregszemlét nyújt a harc kezdete előtt.

```
18.
19.     def egerészik(self):
20.         egér_tapasztalata = random.randint(1, 100)
21.         print(self.név, 'egy', egér_tapasztalata, 'tapasztalatú egérrel próbálkozik.')
22.         if self.tapasztalat > egér_tapasztalata:
23.             print('Megfogta!')
24.             if self.tapasztalat < 100:
```

Aki tapasztaltabb, az nyer.

```
25.                 self.tapasztalat += 1
26.             self.eszik()
27.         else:
28.             print('Elszaladt :(')
```

A sikeres vadászat növeli a tapasztalatot, legfeljebb 100-ig…

…és tömegnövekedést is okoz – egy másik tagfüggvény hívásával.

Kövessük végig valamelyik macskánk egerészéseinek egy hosszabb sorozatát:

```
for _ in range(50):
    print(z.név, 'tömege:', z.tömeg, ', tapasztalata:', z.tapasztalat)
    z.egerészik()
```

Mielőtt az osztály utolsó tagfüggvényének megvalósításába fognánk, itt az ideje szembesülnünk azzal, hogy Pythonban programozva jószerével az első pillanattól kezdve objektumokat használunk. Anélkül, hogy túlzottan mélyen elmerülnénk a témában, lássunk néhány, az előző kijelentést illusztráló programsort, melyeket akár mostani programunk végére is beírhatunk. Megjegyezzük, hogy a „típus” és az „osztály” fogalma többé-kevésbé felcserélhető, például lista típusú adat helyett mondhatunk lista osztályú objektumot is.

A type() függvény megmondja, hogy egy adott objektum melyik objektumosztályba tartozik.

<class '__main__.Macska'> típusú/osztályú objektum

```
print('A z objektum típusa:', type(z))
```

A type() függvény megmondja, hogy egy adott objektum melyik objektumosztályba tartozik.

<class 'int'> típusú/osztályú objektum

```
szám = 5
print('Az 5 típusa:', type(szám))
szám = int(5)
print('Az int(5) típusa:', type(szám))
```

Ilyenkor az int() osztály objektuma az 5 értékkel inicializálódik.

A két sor ugyanolyan listát, ha úgy tetszik, list osztályú objektumot hoz létre.

```
lista1 = [1, 2]
lista2 = list((1, 2))
print('A listák típusai:', type(lista1), type(lista2))

szótár1 = {'kulcs1': 'érték1', 'kulcs2': 'érték2'}
szótár2 = dict(kulcs1 = 'érték1', kulcs2 = 'érték2')
print('A szótárak típusai:', type(szótár1), type(szótár2))
```

A két sor ugyanolyan szótárat, ha úgy tetszik, dict osztályú objektumot hoz létre.

A lecke elején említett definíció második részében („és az adatszerkezeten végezhető műveletek”) arról van szó, hogy milyen műveleteket megvalósító tagfüggvényei vannak egy osztálynak. Ismerjük például a `list` osztály `append()` tagfüggvényét, a `dict` osztály `items()` tagfüggvényét vagy az `str` osztály `strip()`, `split()` és `lower()` tagfüggvényeit.

Utolsó tagfüggvényként egy speciális Python-függvényt valósítunk meg. Pythonban programozva természetesnek vesszük, hogy az egyes objektumoknak van szöveges reprezentációjuk: ki tudjuk adni a `print(szótár1)` parancsot, és értelmezhető kimenetet kapunk, ami különösen a kódolás hibakeresési fázisában hasznos. Mindez saját magunk készítette osztályok objektumaival alapesetben nem működik. Próbáljuk ki!

```
print(z)
```

Nem azt látjuk, aminek igazán örülnénk. Megjelenik ugyan egy, az objektum memóriában való elhelyezkedésével kapcsolatos információ, de jobb lenne, ha látnánk is, hogy mi van az objektumban. Szerencsére mindössze annyi a dolgunk, hogy megvalósítunk egy újabb tagfüggvényt a `Macska` osztályban:

```
29.
30.     def __str__():
31.         return self.név + ' születési éve: ' + str(self.születési_év) + \
32.             ', tömege: ' + str(self.tömeg) + ' g.'
```

Ha egy objektumnak van `__str__()` tagfüggvénye, akkor annak a visszatérési értékét írja ki a `print()` függvény. A Python beépített adattípusai, objektumosztályai mind a fentihez hasonló módon biztosítanak lehetőséget arra, hogy könnyen értelmezhető formában megismerjük az objektum tartalmát. Ha szeretnénk olyan objektumokat készíteni, amelyek minden szituációban a Python beépített objektumaihoz hasonlóan viselkednek, a *magic methods* vagy a *dunder methods* kifejezésre érdemes keresnünk az interneten.

Feladatok

1. Oldjuk meg, hogy macskáink véletlenszerűen nyávogjanak „Nyaú!”-t vagy „Mijaú!”-t! A `Macska.eszik()` tagfüggvény helyett alkossuk meg a `Macska.tápot_eszik()` és a `Macska.egeret_eszik()` tagfüggvényt! Az előző vegye át a régi függvény szerepét, az új pedig legyen paraméterezhető a megevett egér grammban kifejezett tömegével! A megevett egér tömege véletlenszerűen 5–25 százalékban fordítódjék a macska tömegének növelésére! A `Macska.egerészik()` tagfüggvényt módosítsuk úgy, hogy az „egereknek” ne csak tapasztalatuk, hanem véletlenszerű (de a valóságban elképzelhető) tömegük is legyen! Az egér tömegét használjuk fel a `Macska.egeret_eszik()` tagfüggvény hívásakor!
2. Kihívást jelentő feladat: Két macska találkozásának gyakran elkeseredett nyávogás és az egyik fél pánikszerű menekülése a vége. De melyik macska fut el? A megfelelő mágikus tagfüggvények megvalósításával tegyük lehetővé, hogy két `Macska` osztályú objektum a szokásos relációs jelekkel összehasonlítható legyen! Az a `Macska` legyen „nagyobb”, amelyiknek a tapasztalata nagyobb!

Működésben a csoportnapló objektumai

Négy leckével ezelőtt *(Csoportnapló – Tapogatózás az objektumorientált programozás irányába)* írtuk meg a csoportnaplós programunkat először. Akkor még egyáltalán nem használtunk objektumokat, a programunk eljárásközpontú volt. Nem volt rosszabb, mint a programunk második, illetve most elkészülő harmadik változata, de azért az akkori töprengéseink során láttuk, hogy komoly lépéseket kell tennünk, ha azt szeretnénk, hogy a programunk könnyebben áttekinthető, karbantartható és továbbfejleszthető legyen.

A program második változata már használ objektumokat, de ezek az objektumok még csak adatszerkezetek: az adatokkal dolgozó függvények még az objektumokon kívül vannak, azaz még nem tagfüggvények.

Most elkészítjük a program harmadik változatát. Átgondoljuk, hogy melyek azok a függvények, amelyek egy-egy diákkal vagy a diák jegyeivel dolgoznak: azok, amelyek megjelenítik vagy visszatérési értékükben megadják a szóban forgó diák egy-egy jellemzőjét, vagy amelyek műveleteket végeznek a diák valamelyik jellemzőjével.

Az ilyen függvényeket tagfüggvényekké alakítjuk, és megvalósítjuk azokat a lehetőségeket, amelyeket töprengéseink során hasznosnak gondoltunk. A `Diák` osztály jelentősen kibővül:

```
1. class Diák():
2.     def __init__(self, sor):
3.         self.név = sor[0]
4.         self.osztály = sor[1]
5.         jegyek = sor[2].split()
6.         self.rendes_jegyek = []
7.         self.témazáró_jegyek = []
8.         for jegy in jegyek:
9.             if len(jegy) == 1:
10.                self.rendes_jegyek.append(int(jegy))
11.            else:
12.                self.témazáró_jegyek.append(int(jegy[0]))
13.
14.    def évfolyam(self):
15.        return int(self.osztály[:2])
16.
17.    def osztálybetű(self):
18.        return self.osztály[2:]
19.
20.    def osztály(self):
21.        return self.osztály
22.
```

Az __init__() tagfüggvény nem változik.

Ezt a két tagfüggvényt az első töprengésünkben álmodtuk meg.

Ez a tagfüggvény kényelmi szerepű: nem kell azon töprengeni, hogy a diák.osztály (egy jellemző) vagy a diák.osztály() (egy függvény) használatával tudjuk meg a diák osztályát.

```
23.    def megírt_minden_témazárót(self, idei_témazárók_száma):
24.        if self.jegyek_száma('témazáró') == idei_témazárók_száma:
25.            return True
26.        else:
27.            return False
28.
```

Megadhatjuk, hogy hány témazárót írt idén a csoport, így programunk használhatóságát növeljük.

Ha szeretnénk úgy megírni a tagfüggvényt, hogy csak egy `return` utasítás szerepeljen benne, akkor az utolsó négy sorát helyettesítsük ezzel az eggyel:

```
return self.jegyek_száma('témazáró') == idei_témazárók_száma
```

A második töprengésben olyan tagfüggvényt szerettünk volna, amelyiknek a hívási kódját olvasva azonnal látható, hogy mit csinál a program. Az alábbi egy kicsit ügyesebb az akkor elképzeltnél: többféle jegytípust kezel. Ráadásul bővíthető, ha egyszer majd megkülönböztetjük az órai munkára kapott jegyeket. A jegy típusát a `milyen` paraméterben adhatjuk át a tagfüggvénynek.

```
29.     def jegyek_száma(self, milyen):
30.         if milyen == 'témazáró':
31.             return len(self.témazáró_jegyek)
32.         elif milyen == 'rendes':
33.             return len(self.rendes_jegyek)
34.         elif milyen == 'összes':
35.             return len(self.témazáró_jegyek) + len(self.rendes_jegyek)
36.
```

Az utolsó töprengésünknek megfelelően az `átlag()` függvényt úgy alakítjuk át, hogy annak a programozónak, aki használja, ne kelljen tudnia, hogy most épp milyen szabályok szerint számolunk átlagot. Ha az átlagszámítás módja változik, akkor csak ezen az egy helyen kell belenyúlnunk a programba.

```
37.     def átlag(self):
38.         összeg = sum(self.rendes_jegyek) + sum(self.témazáró_jegyek) * 2
39.         osztó = self.jegyek_száma('összes') + self.jegyek_száma('témazáró')
40.         return round(összeg/osztó, 2)
41.
```

Az év végi jegy végső soron a diák tulajdonsága, azaz az objektumosztályban van a helye a jegyet visszaadó függvénynek.

```
42.     def év_végi_jegy(self):
43.         á = self.átlag()
44.         if á <= 1.7:
45.             return 1
46.         elif 1.7 < á < 4.5:
47.             return int(round(á, 0))
48.         else:
49.             return 5
```

A tagfüggvény egy másik tagfüggvényt hív a jegy megállapításakor.

Elkészültünk a `csoportnaplo.py` végső változatával. Persze az objektumorientált programozás témakörét messze nem merítettük ki, szédítő mélységeibe épp csak lepillantottunk. Ahhoz azonban eleget tanultunk, hogy – egy szükséges kitérőt követően – grafikus felhasználói felületű programokat írhassunk.

Egy szükséges vargabetű: kulcsszó-paraméteres függvények és modulok

Ebben a leckében a Python nyelvű programok két olyan, szorosan nem összefüggő részterületével foglalkozunk, amelyek szükségesek ahhoz, hogy értsük a grafikus felhasználói felület programozását. Ugyanakkor mindkét új tudományunk remekül használható a programozás minden más területén is.

Kulcsszó-paraméteres függvények

Futtassuk le az alábbi kódot!

```
1. def köszön(hogyan, kinek):
2.     return hogyan + ' ' + kinek + '!'
3.
4. print(köszön('Ave,', 'Caesar'))
5. print(köszön('Szevasz,', 'Tavasz'))
```

A fenti függvénnyel az a probléma, hogy ha esetleg felcseréljük a paramétereit, akkor nem úgy köszön a programunk, ahogyan szeretnénk. A veszély fokozódik az olyan függvényeknél, amelyek sok paramétert várnak.

A függvények fenti paraméterezése az úgynevezett **helyzeti** vagy pozicionális **paraméterezés**. Azért nevezzük így a módszert, mert a függvény az átadott paraméterek egymáshoz viszonyított helyzetéből, sorrendjéből tudja, hogy melyikkel mi a teendője.

Írjuk át az első sort:

```
1. def köszön(hogyan='Ave,', kinek='Caesar'):
```

Így a függvényünk **kulcsszóparamétereket** (magyarul gyakran fordítjuk nevesített paraméternek is az angol keyword parameter kifejezést) kapott, azaz a paramétereknek kulcsszavuk, nevük van. Az ilyen függvényeknél egyértelmű a paraméterek jelentése, a sorrendjük pedig többé nincs erősen kötve. Ha a kulcsszó-paraméteres függvénynek híváskor nem adjuk meg a paramétereit, akkor a függvénydefinícióban lévő, alapértelmezett paramétert használja a függvény.

Az alábbiakban néhány kódsort, kódsorpárt közlünk, azokat követőleg pedig igaz megállapításokat a kulcsszóparaméterek használatáról. A kódsorok nem minden esetben futnak le helyesen. A feladatunk azt megállapítani, hogy melyik kódrészlet futtatásából melyik megállapítás következik.

```
1)  print(köszön('Szevasz,', 'Tavasz'))
2)  print(köszön())
3)  print(köszön(hogyan='Szevasz,'))
    print(köszön(kinek='Tavasz'))
4)  print(köszön(hogyan='Szevasz,', kinek='Tavasz'))
5)  print(köszön(kinek='Tavasz', 'Ave,'))
6)  print(köszön('Ave,', kinek='Tavasz'))
7)  print(köszön(hogyan='Szevasz,', 'Tavasz'))
```

A) Továbbra is megadhatjuk a paramétereket helyzeti paraméterként.
B) Nem kell minden paramétert megadni. Amelyiket nem adjuk meg, annak az alapértelmezett értékét használja a függvény.
C) Paraméterek nélkül a függvény a definíciókor megadott alapértelmezett értékeket használja.
D) A kulcsszóparaméterek megadási sorrendje felcserélhető.
E) Kulcsszóparamétert nem követhet helyzeti paraméter, még akkor sem, ha jó a paraméterek sorrendje.
F) Kulcsszóparamétert nem követhet helyzeti paraméter.
G) Helyzeti paramétert követhet kulcsszóparaméter, például `print('Szia', 'mia!', sep='-')`

A kulcsszóparaméterek értéke megadható változóval is, így néha az alábbi, első találkozáskor még meglepő függvényhívást látjuk:

```
hogyan = 'Pá,'
kinek = 'kis aranyom'
print(köszön(hogyan=hogyan, kinek=kinek))
```

Ez a paraméter kulcsszava, neve.

Ez a változó neve.

Modulok

Ha eddig szigorúan tartottuk magunkat a tankönyvben írtakhoz, akkor egyetlen modult importáltunk, éspedig a random nevűt. A modul függvényei közül a `randint()` nevűvel véletlen egész számokat állíthatunk elő, a `choice()` pedig arra jó, hogy például a számára átadott lista véletlenszerűen kiválasztott elemét visszaadja.

A modulok arra valók, hogy bizonyos, gyakran használt, előre megírt programrészleteket tároljunk bennük.

Vannak olyanok, amelyek az alap Python-telepítéssel együtt kerülnek a gépünkre, és sok-sok olyan is van, amelyeket külön kell telepítenünk. Az informatika számos – de nem erős túlzás a minden szó használata sem – területével kapcsolatosan léteznek modulok. Van hálózatot kezelő, hangfájlokat tölteni-menteni képes vagy 3D-modellek készítésére alkalmas modul; van olyan, amellyel a közösségi médiára posztolhatunk automatikusan; olyan, amellyel robotot irányíthatunk, játékprogramot fejleszthetünk vagy mesterséges intelligenciát alkothatunk, és még ki tudja, hányféle. Ilyen modulokban kapnak helyet azok az előre elkészített programrészek, amelyekkel a grafikus felhasználói felületű programok írása igen jelentős mértékben egyszerűsödik, és amilyet a következő leckétől kezdve mi is használunk.

A modulok a legegyszerűbb esetben nem mások, mint Python-fájlok, amelyek vagy a fő programfájlunkkal közös mappában, vagy olyan egyéb mappában vannak, ahol a gépünkre telepített Python-parancsértelmező keresi őket. Ezekben a fájlokban nem egész programok találhatók, hanem például változók, függvények vagy osztályok definíciói.

32. példa: Hazánk békafajainak listája

Készítsünk mi is egy modult! Hozzunk létre egy fájlt `beka.py` néven, benne egy, a magyarországi békafajokat felsoroló listával:

```
1. békák = ['vöröshasú unka', 'sárgahasú unka',
2.          'barna ásóbéka', 'barna varangy',
3.          'zöld varangy', 'zöld levelibéka',
4.          'gyepi béka', 'mocsári béka',
5.          'erdei béka', 'tavi béka',
6.          'kis tavibéka', 'kecskebéka']
7.
```

Amikor elkészültünk, nyissunk egy másik fájlt a szerkesztőnkben, és mentsük azt `foprogram.py` néven (a `beka.py`-t még ne zárjuk be)! Az új fájlunknak egyelőre csak három sora lesz:

```
1. import beka
2.
3. print(', '.join(beka.békák))
```

Importáljuk a saját modulunkat.

Az új fájlt mentsük a `beka.py` fájlnéven egy mappába, majd futtassuk!
Ha minden jól ment, azt látjuk, hogy kiíródtak a békafajok nevei. Láttuk, hogy a modulban lévő listára hasonlóan hivatkozunk, mint a `randint()` függvényre szoktunk: a lista neve előtt megadjuk a modul nevét is. Szakkifejezést használva úgy fogalmazunk, hogy a `békák` lista a `beka` *névtérben* van, és az elérésekor megadjuk a névteret is.

Természetesen írhatunk függvényt is a modulunkba. Írjunk egy olyan függvényt, amely a lista valamelyik véletlen elemét adja vissza! Szükség lesz hozzá a random modulra, amelyet kivételesen nem a `beka.py` fájl elején importálunk, mert itt a könyv lapjain utólag nehézkes átszámozni a sorokat. Bővítsük a `beka.py` fájlt:

```
8. import random
9. def véletlen_béka():
10.     return random.choice(békák)
11.
```

Egy modul használhat más modulokat.

Ha mentettük a fájlt, a főprogramban használatba vehetjük új függvényünket, például így:

```
print(beka.véletlen_béka())
```

Hozzunk létre egy osztályt is a modulunkban, a `beka.py` fájlban:

```
12. class Béka():
13.     def __init__(self):
14.         self.faj = véletlen_béka()
15.
16.     def hangot_ad(self):
17.         hang = random.choice(['Brekk!', 'Kurutty!'])
18.         return hang
```

A modulban tudjuk használni a modul egyéb függvényeit.

Az osztályt használatba vehetjük a főprogramból:

```
kedvenc = beka.Béka()
print(kedvenc.faj, 'megszólal:', kedvenc.hangot_ad()
```

Végezetül röviden érintünk még két érdekességet. Tesszük ezt azért, mert az interneten kóborolva mindkettővel gyakran találkozhat a Pythonban programozó programozó, és fontosnak érezzük, hogy értse, amit lát.

Lehetőség van nem teljes modulok, hanem modulrészek importálására is. A

```
from beka import Béka
```

parancs csak a Béka osztályt importálja a modulból, amelyet ezt követően nem `beka.Béka`, hanem egyszerűen `Béka` néven használhatunk. A módszer előnye a rövidebb kód, hátránya, hogy ha a főprogramban is használjuk a Béka nevet, akkor bajba kerülünk, mert az egyik név elfedi, és így használhatatlanná teszi a másikat.

Akár a modulok, akár a modulrészek hivatkozhatók az eredetitől eltérő néven importálást követően. Egész modul esetén

```
import modul as nálunk_használt_név
```

modulrész esetén pedig

```
from modul import rész as nálunk_használt_név
```

a megfelelő utasítás. A módszert olyankor használhatjuk, ha az eredeti név hátráltat bennünket: például használjuk már valami másra, vagy egyszerűen csak túl hosszú.

Grafikus felhasználói felületű alkalmazást fejlesztünk – A Sajáttömb

A programozás témakör hátralévő leckéiben két alkalmazást készítünk el. Mindkettő grafikus felhasználói felülettel (angol szakkifejezéssel: graphical user interface, GUI) rendelkezik.

Hogyan készül egy GUI?

A grafikus alkalmazások fejlesztése az esetek túlnyomó többségében valamilyen mások által elkészített keretrendszer, programrészleteket tartalmazó szoftverkönyvtár használatára támaszkodik. Ezek a keretrendszerek a grafikus felület számos összetevőjét, részét készen bocsátják rendelkezésünkre, azaz nem nekünk kell megírni például azt, hogy

- legyenek ablakaink, amelyek az operációs rendszer grafikus felületével együttműködnek;
- az ablakokon meglegyen a szokásos három gomb (kicsinyítés, teljes képernyő, bezárás);
- az ablakok az egérrel mozgathatók legyenek;
- az ablakok szélét az egérrel megfogva át lehessen őket méretezni;
- és még számos egyéb részt, amelyek közül néhánnyal megismerkedünk alkalmazásaink elkészítése során.

Az egyes keretrendszerek használata, programozása többé-kevésbé hasonló, egyes területeken pedig nagyon eltér egymástól. De ha már bármelyikben programot írunk, akkor elég jó elképzeléseink lesznek a grafikus alkalmazások fejlesztésének folyamatáról, és ezt bármely más környezetben is kamatoztathatjuk.

Vannak olyan keretrendszerek, amelyeket csak egyetlen nyelven fejlesztve tudunk használni, és van, amelyet többből is. Vannak olyanok, amelyek egyetlen operációs rendszeren, grafikus felületen működnek, és vannak, amelyek többfélén.

Pythonban programozva nagyon sok grafikus keretrendszer áll rendelkezésünkre. Az általunk használt **Tk környezet** az alap Python-telepítés részét képezi, a rá támaszkodó programok változtatás nélkül futnak a három nagy asztali operációs rendszeren. Ez a környezet is nagyon **sok konstanst, változót, objektumot, tagfüggvényt tartalmaz,** amelyek nevét, paraméterezését, működését a szakirodalmi leírásokból lehet megismerni. A tárgyalásunkban most ezeket megadjuk.

A Python a Tk környezettel egy erre szolgáló felületen (angolul: interface) kommunikál. A felületet `tkinter` néven modulként importáljuk. A legegyszerűbb grafikus program az alábbi három (öt) sorból áll:

A főciklus

A főciklus minden grafikus felületű program lényegi része. Elindítása előtt kialakítottuk a programunk fő ablakát, de ekkor még nem jelenik meg az ablak. A főciklus elindításakor az ablak megjelenik, és a programunk hozzákezd ahhoz a tevékenységhez, amit a futása alatti idő túlnyomó részében tenni fog: várakozik.

A főciklusban lévő program egy, a gazdája mellett ácsorgó inashoz hasonlítható, aki a gazdája minden szavát figyeli, és ha parancsot kap, rohan végrehajtani. A programunk a „gazdának", azaz a számítógép felhasználójának „parancsairól", billentyűlenyomásairól, egérkattintásairól, azaz a bemeneti perifériákról érkező jelzésekről úgynevezett **események** formájában kap értesítést. A perifériák figyelése az operációs rendszer dolga. Az operációs rendszer veszi észre, ha a felhasználó lenyomott egy gombot, mozgatta az egerét, vagy kattintott vele, illetve az érintőképernyőt használta, és ha a mi programunk van „elöl" (szaknyelven: fókuszban), akkor továbbítja neki az eseményt. A programunk pedig úgy reagál az eseményre, ahogy megírjuk a programkódban.

A Sajáttömb

Az első alkalmazásunk egy, a mindenki által ismert Jegyzettömbhöz hasonló program, a *Sajáttömb* lesz. Írjuk be a fenti három kódsort egy `sajattomb` nevű programba, és futtassuk a programot – akár a fájlkezelőből – dupla kattintással! (Windowson a grafikus felülettel működő Python programoknak érdemes `.pyw` kiterjesztést adni.)

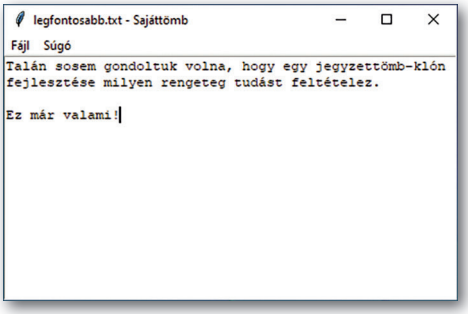
Látjuk, hogy a programunknak van ablaka, rendelkezik címsorral, az azon lévő nyomógombokkal, és átméretezhető akár a gombok használatával, akár az ablak szélét egérrel megfogva. Mindezekért egyáltalán nem dolgoztunk meg a kódunkban – valaki más, a modul írója dolgozott helyettünk. Szempontunkból a programunk „magától" képes ezekre a mutatványokra. Talán ezen a ponton kezdjük el igazán megérteni, hogy milyen előnyökkel jár a grafikus keretrendszerek használata, illetve azt, hogy egy-egy modul betöltése miként növeli ugrásszerűen a lehetőségeinket.

▶ A Sajáttömb Linuxon

Ha a programunkban van a főciklusba való belépést követő utasítás is, az egészen addig nem fut le, amíg a főablak be nem zárul.

Próbáljuk ki: írjunk ki egy szót a szokásos `print()` függvénnyel a főciklus sora után! A programot ezúttal ismét érdemes parancssorból futtatni. A fejlesztés alatt egyébiránt minden grafikus programot érdemes a parancssorból futtatni, mert itt látjuk majd a program hibaüzeneteit is.

A grafikus alkalmazások fejlesztése során nem ritka, hogy az alkalmazás egy önálló osztályba kerül, és a programunk lényegében az osztályból példányosított egyetlen objektumként fut. Ennek többek között az objektumjellemzők



▶ A Sajáttömb Windowson

egyszerűbb elérése az oka: az objektum tagfüggvényei az objektumjellemzőket akkor is egyszerűen tudják módosítani, ha azok a függvényen kívül is léteznek. Mi is kihasználjuk ezt az előnyt. Alakítsuk ilyenné a programunkat:

```
1. import tkinter
2.
3. class App:
4.     def __init__(self, master):
5.         self.master = master
6.
7. főablak = tkinter.Tk()
8. app = App(főablak)
9.
10. főablak.mainloop()
```

Lényegében az egész program itt kap helyet.

Az objektum megyjegyzi, hogy ki a „főnök”, azaz hogy melyik ablak belsejében fut. Ez később még jól jöhet. A paraméter neve hagyományosan master, de átnevezhetjük.

Itt van az egyetlen példányosítása az osztálynak. A paraméterben megmondjuk, hogy a főablak belsejében fusson a program.

A programunkat ezen a ponton lefuttatva nem sok változást látunk, de a főablakban már fut a programunk lényegi része – csak még nem csinál semmit.

A szövegmező

A programunk írását azzal folytatjuk, hogy elkészítjük és megjelenítjük a jegyzettömbünk lényegi részét, azt a szövegmezőt, ahová a szöveget lehet írni. Tapasztalatból tudjuk, hogy az ilyen szövegmező sok mindenre képes. Lássunk néhányat a képességei közül:

- Kurzorral jelzi, hogy hova kerül a következő karakter. A kurzor helyzetét a tevékenységeinknek megfelelően frissíti.
- Ha a billentyűzetről karaktert kap (egy *esemény* történik), akkor a karaktert megjeleníti.
- Enter lenyomására új sort kezd, a kurzort a sor elejére teszi.
- A sor végét elérve új sort kezd.
- A szokásos módon tudunk benne másolni és beilleszteni.

Mindezzel nagyon kevés dolgunk lesz, hiszen a keretrendszer megfelelő osztálya – a `Text` – már készen áll, nekünk csak példányosítani kell, és elhelyezni a példányt. Erre való az `App` osztályunkban elhelyezett alábbi tagfüggvény:

A Tk keretrendszer Text osztályából példányosítunk egyet magunknak.

```
    def szövegterületet_készít(self):
        self.szövegterület = tkinter.Text(self.master)
        self.szövegterület.pack(expand=True, fill='both')
```

A példányosításkor megmondjuk, hogy melyik ablakba kerüljön a szövegmező – ezért tettük el a főablakhoz vezető hivatkozást.

A pack() tagfüggvény hívása az egyik módszer egy grafikus objektum megjelenítésére a tkinterben. A paraméterei szerint nőjön akkorára, amekkorára lehet (expand), mindkét dimenzióban (both).

A fenti tagfüggvény hívására minden induláskor szükség van, úgyhogy helyezzük el az `__init__()` tagfüggvény végén ezt a sort:

```
self.szövegterületet_készít()
```

És indíthatjuk a programunkat. Igazán remekül működik!

A menüsor

A menüsorban két választási lehetőség lesz: a Fájl, illetve a Súgó. A menüket kialakító tagfüggvény:

```
def menüsort_készít(self):
    self.menüsor = tkinter.Menu(self.master)

    fájl_menü = tkinter.Menu(self.menüsor, tearoff=0)
    fájl_menü.add_command(label='Új')
    fájl_menü.add_command(label='Megnyitás')
    fájl_menü.add_command(label='Mentés')
    fájl_menü.add_separator()
    fájl_menü.add_command(label='Beállítások')
    fájl_menü.add_separator()
    fájl_menü.add_command(label='Kilépés', command=self.master.destroy)
    self.menüsor.add_cascade(label='Fájl', menu=fájl_menü)

    súgó_menü = tkinter.Menu(self.menüsor, tearoff=0)
    súgó_menü.add_command(label='Névjegy')
    self.menüsor.add_cascade(label='Súgó', menu=súgó_menü)

    self.master.config(menu=self.menüsor)
```

A Tk keretrendszer Menu osztályából példányosítunk egyet magunknak.

Létrehozzuk a Fájl menüt.

Összeállítjuk a Fájl menü tartalmát.

Elhelyezzük menüt a menüsorban.

Összeállítjuk a Súgó menüt.

Elhelyezzük a menüsorban.

A menüt nem lehet leválasztani a menüsorról.

Természetesen szükség van a tagfüggvény hívására az `__init__()` metódus végén:

```
self.menüsort_készít()
```

A programot elindítva látjuk, hogy a menü megjelenik, de nem működik, a *Kilépés* menüpontot kivéve. A kódot újra átnézve látjuk, hogy ennek az egy menüpontnak adtuk meg a `command`, azaz parancs paraméterét, ezért épp ez az egy működik. A menüpont parancsa a főablak `destroy()`, azaz 'megsemmisít' tagfüggvénye – ez fut le, ha a *Kilépés* menüpontot választjuk. A végrehajtandó függvényeket a `command` paraméterben a függvény végi zárójel nélkül kell megadnunk – azaz nem tudunk számukra paramétert átadni, és ebből még lesz problémánk.

A névjegy

Sok „igazi" programhoz hasonlóan a *Sajáttömb* is kap névjegyet, ahol például a program készítőit szokás megemlíteni. A névjegy új ablakának megtervezésével nem óhajtunk sok időt tölteni, egy rendelkezésre álló, úgynevezett üzenetablakot használunk.

Az első dolgunk a `tkinter` modulcsomag egy újabb moduljának importálása. A következő sort írjuk a meglévő `import` alá:

```
import tkinter.messagebox
```

Helyezzük el az osztályban az alábbi tagfüggvényt:

```
def ablak_névjegy(self):
    tkinter.messagebox.showinfo('Névjegy', 'Mi írtuk az egészet!')
```

Végül módosítsuk a *Súgó* menü megfelelő sorát, hogy futtassa az előbb elkészített tagfüggvényt:

```
súgó_menü.add_command(label='Névjegy', command=self.ablak_névjegy)
```

Vezérlőelemek, widgetek

Látjuk, hogy nagyon leegyszerűsítve egy grafikus felhasználói felület kétféle összetevőből áll: ablakokból és widgetekből (ejtsd: vidzsit). A widget szó a window + gadget (ablak + bigyó, esetleg ablak + kütyü) összevonás eredménye. Eddig két widgetet használtunk: a szövegmezőt `(Text)` és a menüsort `(Menu)`. További widgetek például a nyomógombok, a rádiógombok, a jelölőnégyzetek, a keretek, a gördítősávok és a legördülő menük; néhánnyal még megismerkedünk a könyvünkben.

A widgeteket szokás controlnak (magyarul vezérlőelemnek vagy csak vezérlőnek) is nevezni. Az, hogy melyik elnevezést használjuk, a szóban forgó keretrendszertől függ.

A grafikus programok eleje és vége

Mostanra látjuk, hogy egy grafikus programnak nincs eleje és vége abban az értelemben, ahogy az eddig írt programjainknak volt. Nem tudjuk megmutatni, hogy milyen sorrendben fognak lefutni az utasítások. Ha a *Sajáttömb* szövegterületére írunk, akkor az a widget aktív, ha a menüben bóklászunk, akkor pedig az. A program lényegében sok objektum együttese, melyek között mi teremtjük meg a kapcsolatot, és amelyek többnyire aszerint dolgoznak, hogy melyik milyen eseményről szerez tudomást a főciklus ismétlődései során.

A Sajáttömb fájlkezelése

Alkalmassá kell tennünk a programunkat a szerkesztett fájl mentésére, létező fájl megnyitására és új fájl kezdésére. Kezdjük ez utóbbival!

A program két esetben hoz létre új fájlt: az alkalmazás megnyitásakor, és amikor a megfelelő menüpontot választjuk. Lássuk azt a tagfüggvényt, amelyik mindkét esetben hívható:

Ha volt már ilyen változónk, ürítjük a tartalmát. Ha nem volt, mert most indult a program, akkor most létrehozzuk.

```
def fájl_üreset_állít(self):
    self.fájlnév = None
    self.szövegterület.delete('1.0', 'end')
```

Az első sor nulladik karakterétől a szövegterület végéig törlünk.

Helyezzük el a tagfüggvényt hívó sort az `__init__()` metódus végén, és adjuk meg az *Új* menü paramétereként is.

A második fájlkezelő tagfüggvényünk a fájl mentésére való. Mi is lesz a dolga?

- Ha még nincs fájlnév, akkor új fájlnevet kér.
- Ha van már fájlnév, azaz nem ez az első mentés, akkor azt használja.
- A fájlnévvel menti a szövegterület tartalmát – új fájlt ír, vagy felülírja az előző fájlt.

A programunknak nem lesz „mentés másként” lehetősége.

Az első dolgunk egy fájlválasztó párbeszédablak importálása az `import tkinter.filedialog` paranccsal.

A mentést végző tagfüggvény:

Ha még sosem volt mentve a fájl...

...akkor párbeszédablakot nyitunk, és bekérjük a fájlnevet.

Ha már volt mentve a fájl, akkor a meglévő nevet használjuk.

Ez az ellenőrzés azért kell, mert ha a párbeszédablakban Esc-et nyom a felhasználó, üres karakter lesz a fájlnév. Így Esc esetén tétlenek maradunk.

A célfájlba a szövegterület tartalma kerül.

```
def fájl_ment(self):
    if not self.fájlnév:
        fájlnév = tkinter.filedialog.asksaveasfilename(
                initialfile='szöveg.txt',
                defaultextension='.txt',
                filetypes=[('Szövegfájlok', '*.txt'),
                           ('Minden fájl', '*.*')])
    else:
        fájlnév = self.fájlnév
    if fájlnév != '':
        with open(fájlnév, 'w') as célfájl:
            célfájl.write(self.szövegterület.get('1.0', 'end'))
        self.fájlnév = fájlnév
```

Adjuk meg a tagfüggvényt a megfelelő menüpont `command` paramétereként, és teszteljük a programunkat!

A fájl megnyitása hasonlít a betöltéshez. Ugyanazt a párbeszédablakot használjuk.

Törlünk, majd betesszük a fájl tartalmát.

```
def fájl_megnyit(self):
    fájlnév = tkinter.filedialog.askopenfilename(
                defaultextension='.txt',
                filetypes=[('Szövegfájlok', '*.txt'),
                           ('Minden fájl', '*.*')])
    if fájlnév != '':
        with open(fájlnév) as forrásfájl:
            self.szövegterület.delete('1.0', 'end')
            self.szövegterület.insert('1.0', forrásfájl.read())
        self.fájlnév = fájlnév
```

A címsor

A programok címsorában szokás megadni az épp szerkesztett fájl nevét és az alkalmazás nevét. Mi is így teszünk, egy újabb tagfüggvénnyel. A tagfüggvény az `os` modulra támaszkodik a fájlnév és az elérési út szétválasztásakor, ne feledjük ezt a modult is importálni!

```
def ablakcímet_állít(self):
    if self.fájlnév:
        cím = os.path.basename(self.fájlnév)
    else:
        cím = 'névtelen'
    cím = cím + ' - Sajáttömb'
    self.master.title(cím)
```

A főablak címsorát állítjuk.

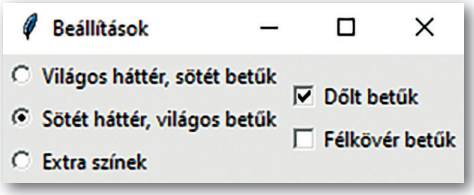
Helyezzük el az új tagfüggvény hívását mindhárom, fájlokkal foglalkozó metódusunk legvégére!

Ha minden jól ment, remekül működik az első grafikus felületű alkalmazásunk – kivéve a *Beállítások* menüt, amely külön leckét érdemel.

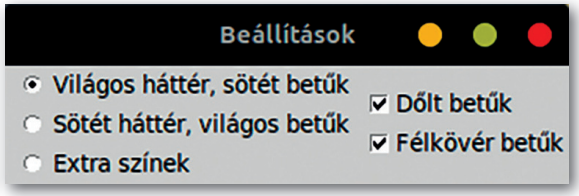
A Sajáttömb beállításai

Ebben a leckében a *Sajáttömb* egy új ablakot kap, amelyben a szövegterület színsémája, illetve a szövegterületen használt betűkészlet jellemzői állíthatók. A *Beállítások* ablakot az eddigieknek megfelelően, az `App` osztály tagfüggvényeként valósítjuk meg.

Két új widget



▶ A Beállítások ablak Windowson



▶ A Beállítások ablak Linuxon

Látjuk, hogy az ablakban rádiógombokkal oldottuk meg a színsémák kiválasztását. **Rádiógombokat** olyankor használunk, amikor a felkínált lehetőségek közül csak egy választható – ahogy a rádión is egyszerre egy csatornát tudunk csak hallgatni. (A mai rádióknak persze a legritkább esetben néznek ki így a gombjaik – ha egyáltalán vannak –, de amikor a rádiógombok megjelentek a GUI-kon, még könnyebben értelmezhető volt a név.)

A rádiógombok egy csoportot alkotnak. Úgy különülnek el a betűkészlet beállításaival foglalkozó jelölőnégyzetektől, hogy az ablakban egymás mellett két **keretet** (frame) helyeztünk el. A kereteket azért nem látjuk, mert létrehozásukkor nem intézkedtünk a hátterükként szolgáló ablaktól való elkülönülésükről. A bal oldali keretbe kerültek a rádiógombok, a jobb oldaliba a jelölőnégyzetek.

Jelölőnégyzeteket olyankor használunk, ha a választható lehetőségek külön-külön is bekapcsolhatók. A jelölőnégyzeteket egyre fokozódó ütemben cserélik az alkalmazások tervezői az érintőképernyővel jobban használható – és így a felhasználók számára a telefonjaikról ismerősebb – **csúszkákra.**

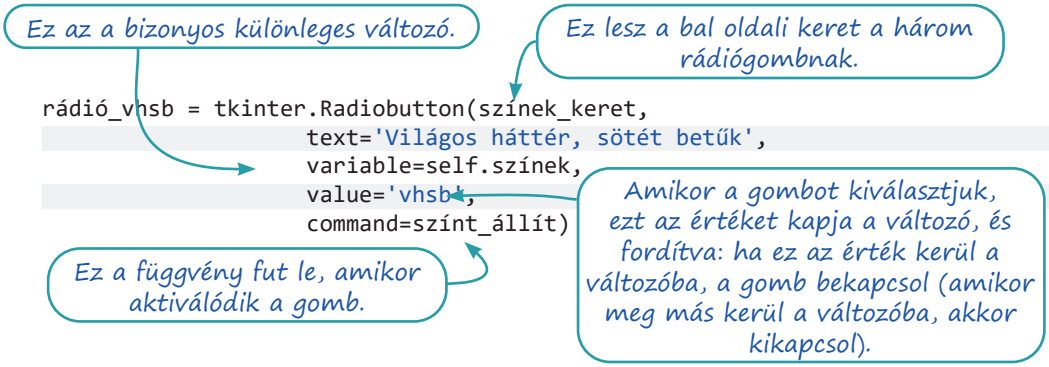
A rádiógombjaink

A három rádiógomb önálló widget, önálló `Radiobutton` osztályú objektum. Mégis valamiként értesülniük kell egymás állapotáról, hiszen ha az egyiket bekapcsoljuk, annak, amelyik eddig be volt kapcsolva, automatikusan ki kell kapcsolnia. Ráadásul a ki-be kapcsolásnak működnie kell akkor is, ha nem a felhasználó kapcsolta át a beállítást másik gombra, hanem a programunk „egyedül” tette ezt meg valamilyen okból, a felhasználó közreműködése nélkül. Bár most nem fogjuk megírni, de például elképzelhető, hogy induláskor betöltjük az utolsóként használt színsémát, és a megfelelő rádiógombot is aktiváljuk.

A gombok egy, a létrehozásukkor paraméterként megadott, közös változó segítségével tartják egymással a kapcsolatot. Létrehozásukkor megadjuk azt is, hogy a változó mely értéke esetén kapcsolódjanak be, minden más érték esetén kikapcsolva maradnak.

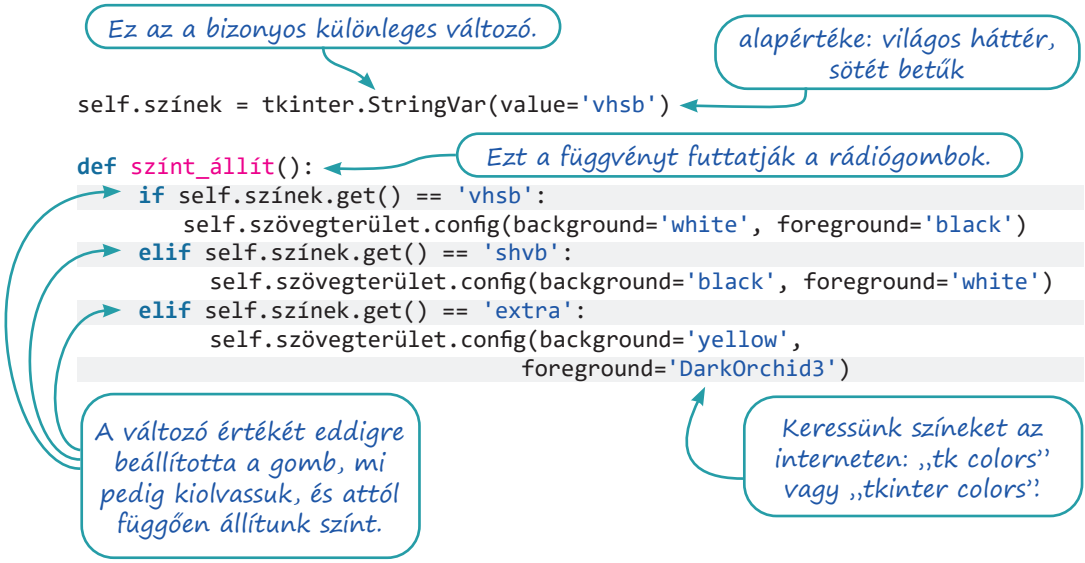
Ez a változó különleges változó, mert az értékének változásakor létrejövő **eseményről** a hozzá kapcsolódó gombok értesülnek. A gombok kódját pedig úgy írták meg, hogy az esemény bekövetkeztekor vizsgálják felül saját állapotukat, és szükség esetén változtassák meg.

Lássuk az első rádiógombunkat kialakító kódot:



Ilyen gombból helyezünk el hármat, csak a másik kettőnek más a neve, és más a kiírt szövege. Mindhármuknak ugyanaz a változója (a `színek`), de más-más értéket kapnak.

A lefuttatandó függvénynek nem tudunk paramétert átadni, és ahogy azt már az előző leckében előrevetítettük, ez most problémát jelent. Más-más színsémát kell beállítani a három gombnak, tehát vagy három külön tagfüggvényt írunk (ez a kód karbantarthatósága miatt nem jó ötlet), vagy helyettesítenünk kell a függvény paraméterezését. Íme a megoldás:



Látható, hogy sikerült olyan megoldást találnunk, amellyel nagyon könnyen vehetők fel új színek. Mindössze annyi a dolgunk, hogy elhelyezünk újabb rádiógombokat, és újabb `elif`-ágakkal bővítjük az `App.szint_állít()` tagfüggvényt.

A jelölőnégyzeteink

Míg a három rádiógombnak össze kellett dolgoznia, a jelölőnégyzeteink (`Checkbutton` osztályú objektumok) külön életet élnek: a betűk dőltsége nem függ össze a betűk vastagságával. Mégis szükség lesz az előzőhöz hasonló különleges változókra – jelölőnégyze-

tenként egyre. Azért kellenek, mert tárolnunk kell, hogy a jelölő épp be vagy ki van-e kapcsolva, illetve az érdemi munkát végző, a betűk tulajdonságát állító függvénynek most sem tudunk paramétert átadni, azaz ebből a változóból tudja meg, hogy ki- vagy bekapcsoltuk-e a jelölőt. Ezek a változók egész számot tárolnak, a 0 a kikapcsolt és az 1 a bekapcsolt állapotot jelenti.

A Beállítások ablak megvalósítása

A betűk tulajdonságainak állítása miatt szükség lesz a `tkinter.font` modulra, ezért importáljuk. Helyezzük el a három változónkat az `__init__()` tagfüggvény végén:

```
self.színek = tkinter.StringVar(value='vhsb')
self.dőlt = tkinter.IntVar(value=0)
self.félkövér = tkinter.IntVar(value=0)
```

Maga a tagfüggvény pedig a következő:

```
def ablak_beállítások(self):

    def színt_állít():
        if self.színek.get() == 'vhsb':
            self.szövegterület.config(background='white', foreground='black')
        elif self.színek.get() == 'shvb':
            self.szövegterület.config(background='black', foreground='white')
        elif self.színek.get() == 'extra':
            self.szövegterület.config(background='yellow',
                                      foreground='DarkOrchid3')

    def dőltet_kapcsol():
        betű = tkinter.font.Font(font=self.szövegterület.cget('font'))
        if self.dőlt.get() == 1:
            betű.configure(slant=tkinter.font.ITALIC)
        elif self.dőlt.get() == 0:
            betű.configure(slant=tkinter.font.ROMAN)
        self.szövegterület.config(font=betű)

    def félkövéret_kapcsol():
        betű = tkinter.font.Font(font=self.szövegterület.cget('font'))
        if self.félkövér.get() == 1:
            betű.configure(weight=tkinter.font.BOLD)
        elif self.félkövér.get() == 0:
            betű.configure(weight=tkinter.font.NORMAL)
        self.szövegterület.config(font=betű)
```

Ezt a függvényt már ismerjük.

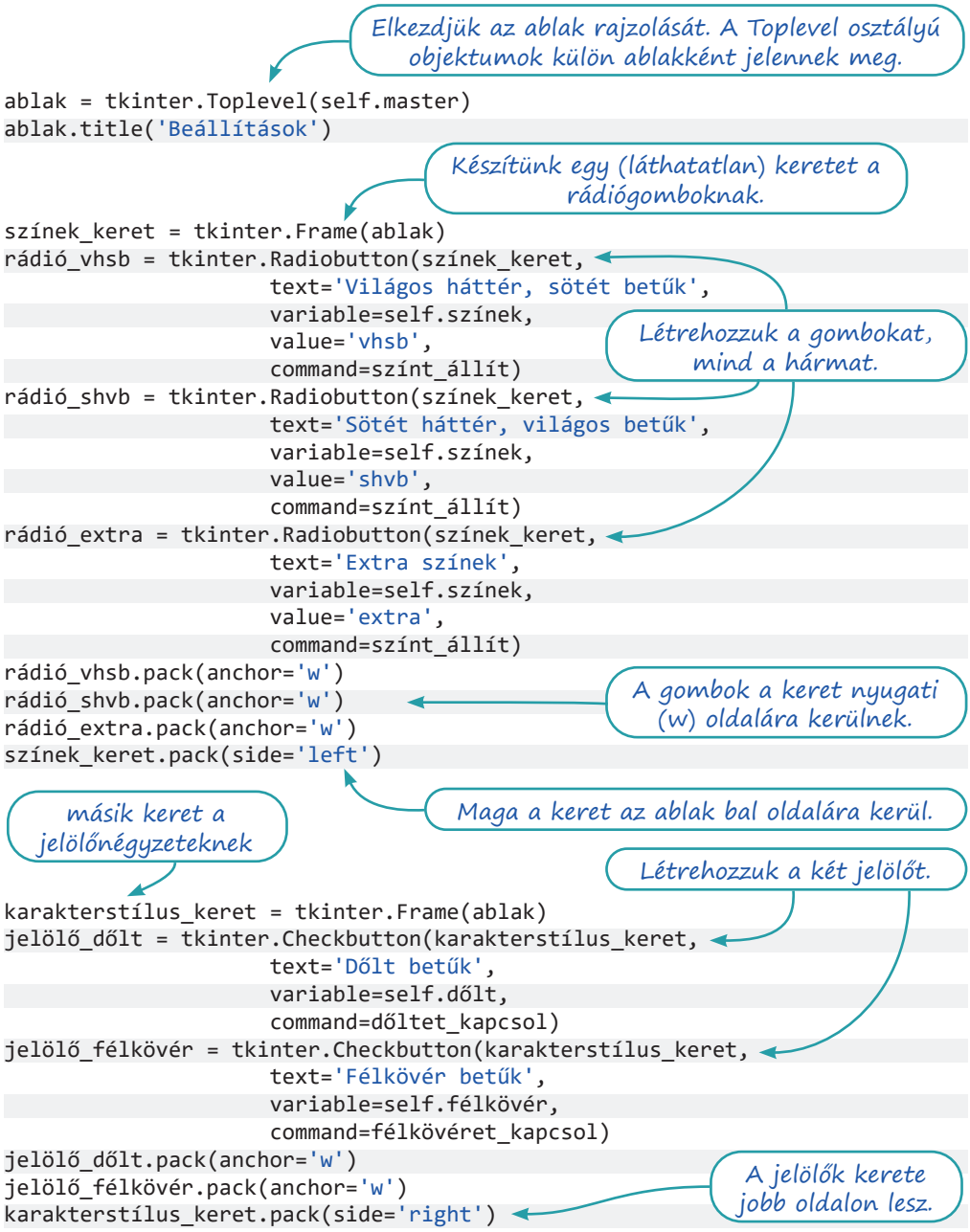
Ez a függvény kapcsolja ki-be a betűk dőltségét.

Kiolvassuk, hogy most milyen a betű (típus, méret).

A self.dőlt változó értékétől függően állítunk dőltet vagy egyeneset. A típus és a méret olyan lesz, amilyet kiolvastunk.

Ez a függvény állítja a betűk vastagságát.

A self.félkövér változó értékétől függően állítunk vastagabbat vagy vékonyabbat.

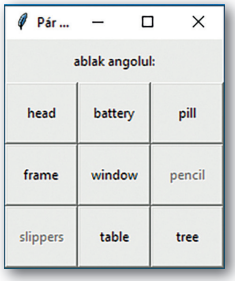



```
ablak = tkinter.Toplevel(self.master)
ablak.title('Beállítások')

színek_keret = tkinter.Frame(ablak)
rádió_vhsb = tkinter.Radiobutton(színek_keret,
                      text='Világos háttér, sötét betűk',
                      variable=self.színek,
                      value='vhsb',
                      command=színt_állít)
rádió_shvb = tkinter.Radiobutton(színek_keret,
                      text='Sötét háttér, világos betűk',
                      variable=self.színek,
                      value='shvb',
                      command=színt_állít)
rádió_extra = tkinter.Radiobutton(színek_keret,
                      text='Extra színek',
                      variable=self.színek,
                      value='extra',
                      command=színt_állít)
rádió_vhsb.pack(anchor='w')
rádió_shvb.pack(anchor='w')
rádió_extra.pack(anchor='w')
színek_keret.pack(side='left')

karakterstílus_keret = tkinter.Frame(ablak)
jelölő_dőlt = tkinter.Checkbutton(karakterstílus_keret,
                      text='Dőlt betűk',
                      variable=self.dőlt,
                      command=dőltet_kapcsol)
jelölő_félkövér = tkinter.Checkbutton(karakterstílus_keret,
                      text='Félkövér betűk',
                      variable=self.félkövér,
                      command=félkövéret_kapcsol)
jelölő_dőlt.pack(anchor='w')
jelölő_félkövér.pack(anchor='w')
karakterstílus_keret.pack(side='right')
```

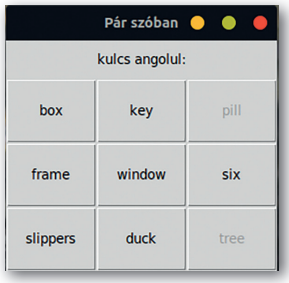
Utolsó feladatunk a tagfüggvény hívásának beállítása a megfelelő menüpont helyes paraméterezésével. Ha minden rendben működik, bátran kiörvendezhetjük magunkat az első „igazi” alkalmazásunk elkészülte alkalmából!

Grafikus felhasználói felületű alkalmazást fejlesztünk – A Pár szóban

A *Pár szóban* egy szótanuló alkalmazás. Induláskor beolvas egy szövegfájlt, benne soronként egy magyar szóval és az angol megfelelőjével. Kiválaszt közülük kilencet, és egyetlen szótárként (a magyar szavak a kulcsok) átadja őket az alábbi ablaknak:



► A Pár szóban Windowson



► A Pár szóban Linuxon

Az alkalmazás ablakának kialakítása

Az alkalmazás kiválaszt egyet a szótár elemei közül, amelyre vonatkozóan felteszi az ablak felső részén látható kérdést. A szótár értékei alapján létrehozza a kilenc gombot, melyeket soronként hármasával elhelyez az ablak alsó részén. Az egyes gombok neve megegyezik a rajtuk olvasható angol szóval.

A kérdezett szó megjelenítésére **címkét,** azaz `Label` osztályú objektumot használunk. Olyan pár szavas, pár mondatos szövegek megjelenítésére szolgál, amelyeket a felhasználó nem tud átírni.

Egy-egy widget elhelyezésére eddig a `tkinter` osztályainak `pack()` tagfüggvényét használtuk, amely elég egyszerűen működik ahhoz, hogy különösebb magyarázat nélkül is boldoguljunk vele. Ezúttal a bonyolultabb elrendezéseket lehetővé tévő `grid()` tagfüggvényt is bevetjük. Használatakor sorok és oszlopok jönnek létre az ablakon vagy kereten belül, és meg kell adnunk, hogy melyik oszlopba és melyik sorba kerüljön a widget. Grid, azaz **rács** használatával jelenítjük meg a `Button` osztályból példányosított gombokat.

Az eddig leírt működést megvalósíthatjuk az alábbi kóddal:

```
 1. import tkinter
 2. import random
 3.
 4. def fájlt_beolvas():
 5.     szójegyzék = {}
 6.     with open('szavak.txt') as forrásfájl:
 7.         for sor in forrásfájl:
 8.             magyar, angol = sor.strip().split()
 9.             szójegyzék[magyar] = angol
10.     megmaradó_szavak = random.sample(list(szójegyzék), 9)
11.     szójegyzék = { kulcs: érték for kulcs, érték in
12.                    szójegyzék.items() if kulcs in megmaradó_szavak }
13.     return szójegyzék
14.
```

Külön függvényként, valósítjuk meg, mert eléggé elkülönül a lényegi résztől.

Kiválasztunk kilenc szót, amelyeket megtartunk.

Szótárértelmezés. Megvalósítható hagyományos ciklusként is.

```
15. class App():
16.     def __init__(self, master, szójegyzék):
17.         self.szójegyzék = szójegyzék
18.         self.keresett_szó = random.choice(list(self.szójegyzék.keys()))
19.         self.kérdés_keret = tkinter.Frame(master)
20.         self.kérdés_keret.pack()
21.         self.kérdés = tkinter.Label(self.kérdés_keret,
22.                                     text=self.keresett_szó + ' angolul:',
23.                                     pady=10)
24.         self.kérdés.pack()
25.         self.gombok_keret = tkinter.Frame(master)
26.         self.gombok_keret.pack()
27.         self.gombokat_készít()
28.         self.gombokat_megjelenít()
29.
30.     def gombokat_készít(self):
31.         self.gombok = {}
32.         for angol in self.szójegyzék.values():
33.             self.gombok[angol] = tkinter.Button(self.gombok_keret,
34.                                                 text=angol,
35.                                                 height=3, width=8)
36.
37.     def gombokat_megjelenít(self):
38.         megjelenítendő_gombok = list(self.gombok)
39.         for sor in range(3):
40.             for oszlop in range(3):
41.                 aktuális_gomb = megjelenítendő_gombok.pop()
42.                 self.gombok[aktuális_gomb].grid(row=sor, column=oszlop)
43.
44. főablak = tkinter.Tk()
45. főablak.title('Pár szóban')
46. app = App(főablak, fájlt_beolvas())
47. főablak.mainloop()
```

Az első keresett szó.

Két keretünk lesz egymás fölött. A felsőbe kerül a kérdés, az alsóba a gombok.

A kérdés rákerül a címkére.

Előbb elkészítjük mind a kilenc gombot…

…aztán kitesszük őket.

A gombok egy szótárba kerülnek.

Az angol szó kerül rájuk.

Ekkorák lesznek.

Itt készül a 3×3-as formáció.

A gombot a pop() kiveszi a még megjelenítendő gombok közül, és értékül adja az aktuális_gomb változónak.

Itt adjuk át a lényegi résznek a kiválasztott szópárokat.

Az alkalmazás működni kezd

Két tagfüggvénnyel bővítjük az alkalmazás `App` osztályát. Az első a gombok lenyomásakor összehasonlítja a gomb nevét (azaz a rajta lévő angol szót) a felső részen olvasható szó angol megfelelőjével. Ha azonosak, akkor kikapcsoljuk a gombot, hogy ne lehessen újra lenyomni. A másikkal végignézzük a szójegyzékünket, illetve azokat a gombokat, amelyek még lenyomhatók, és összevetve őket új szót írunk ki felülre.

Persze a két tagfüggvénynek csak akkor van értelme, ha hívjuk is őket. Az elsőt, az egyezést vizsgálót kell hívnunk a megfelelő gomb lenyomásakor, és ha jó gombot nyomott a felhasználó, akkor majd ez a függvény hívja a másikat, amelyik kiírja az új kérdést.

A függvényhívást a gombok `command` paraméteréül kell megadnunk a 35. sor zárójelén belül. Természetesen most sem tudunk paramétert átadni a függvénynek, pedig jó volna az egyezést vizsgáló függvény tudomására hozni, hogy melyik gombot nyomta le a felhasználó. Az előző leckében megismert különleges változóink helyett ezúttal a korábban futólag említett lambda-függvényen alapuló trükkel leszünk úrrá a problémán. A paraméter:

```
command=lambda angol_szó=angol: self.egyezést_vizsgál(angol_szó)
```

A két tagfüggvény:

Ezt a szót kaptuk paraméterül a lenyomott gombtól.

Ha megegyezik a felülre kiírt szóval...

```
44.     def egyezést_vizsgál(self, szó):
45.         if szó == self.szójegyzék[self.keresett_szó]:
46.                 self.gombok[szó].config(state='disabled')
47.                 self.keresett_szót_frissít()
48.
```

...akkor kikapcsoljuk a gombot, és hívjuk az új kérdést feltevő függvényt.

A még ki nem kapcsolt gombokon lévő szavakat kérdezhetjük.

```
49.     def keresett_szót_frissít(self):
50.         még_nem_kérdezett_szavak = []
51.         for magyar, angol in self.szójegyzék.items():
52.             if self.gombok[angol]['state'] != 'disabled':
53.                 még_nem_kérdezett_szavak.append(magyar)
```

Ha találtunk még kérdezhető szót...

...akkor kérdezünk...

```
54.         if még_nem_kérdezett_szavak:
55.             self.keresett_szó = random.choice(még_nem_kérdezett_szavak)
56.             self.kérdés.configure(text=self.keresett_szó + ' angolul:')
57.         else:
58.             self.kérdés.configure(text='Kifogytam a szavakból.')
```

*...különben **elbúcsúzunk**.*

Elérve a tankönyvsorozat programozásról szóló leckéi közül az utolsó végére, remélhetőleg senki számára sem teljes boszorkányság többé a számítógép programjainak működése. Sokkal pontosabb elképzeléseink lettek a bennünket mindennapjainkban körülvevő eszközök belső működéséről. Alighanem mindenki előtt, aki eddig velünk tartott, kitágult a horizont. Lesznek olyanok, akik már ezt is túlzásnak vélik, másokat pedig újabb, hosszabb felfedezőutakra csábít az, amit eddig megismertek csodálatos világunknak ebből a részéből.

Szerencsés utat!

D A digitális eszközök használata

„Ha egy fészekben van 10 kakukktojás, és egy fürjtojás, akkor melyik a kakukktojás?”

Informatikai hálózatok és felhőszolgáltatások

Ez a fejezet – a kilencedikes és tizedikes tankönyv *A digitális eszközök használata* című fejezetéhez hasonlóan – egy kakukktojás a könyvben. Az itt leírtak nem önálló tanulási egységei a tananyagnak, hanem azokat az eszközhasználattal kapcsolatos tudnivalókat foglalják össze, amelyek a többi fejezetben a téma kapcsán eddig felmerültek. A tankönyvsorozat előző két kötetében a digitális eszközhasználat sok olyan területéről szó esett már, amelyekre lehet, hogy most lenne szükségünk, ezért érdemes a tizedikes tankönyv azonos című fejezetét is fellapozni, ha ebben a témakörben keresünk valamit. Ebben a rendhagyó fejezetben elsősorban az informatikai hálózatról, az abban használt eszközökről és feladataikról, valamint a hálózaton keresztül elérhető szolgáltatásokról lesz szó. Ha valamelyik téma felkeltette az érdeklődésünket, akkor az itt leírtaknak érdemes lehet máshol is utánanézni.

Mivel az itt ismertetett tudnivalók nem kötelező részei a tananyagnak, a fejezetet záró kérdések megválaszolása során szintén szükség lehet internetes kutatásra. Sok érdekességre lelhetünk, míg megtaláljuk a válaszokat.

Számítógépes hálózatok

A célok és a történelem

A számítógépes hálózatok kialakítását az 1960-as években kezdték, és az elsődleges cél a számítógépes utasítások továbbítása volt a gépek között. Az internet alapjait már az 1970-es évek első felében letették, a ma is használt TCP/IP-szabvány szerinti hálózatok tesztelése már ekkor elkezdődött. Az internetet sokan a széles körben elterjedten használt világhálóval (angol eredetiben: World Wide Web, WWW vagy röviden web) azonosítják, amely a http (HyperText Transfer Protocol) segítségével teszi lehetővé, hogy böngészőprogramon keresztül jussunk információhoz. Az internet ennél jóval több, elég, ha csak arra gondolunk, hogy elektronikus leveleket küldhetünk, fogadhatunk, cseveghetünk, vagy akár videókonferencián vehetünk részt, bár sok esetben ezeket is a böngésző segítségével tesszük.

A számítógépes hálózatok már az otthonunkon belül is megjelentek. Teljesen természetes, hogy több digitális eszköz kapcsolódik az internetre, nemcsak a számítógépünk, hanem a telefonunk vagy akár a tabletünk is. Egyre több a háztartásokban megjelenő **okoseszköz.** Nézzünk például egy okostermosztátot, amelynek feladata a lakás fűtésének szabályozása. A kis készülék akár távolról is vezérelhető, azaz egy hosszabb utazás végén a telefonunkról magasabbra állíthatjuk a fűtési hőmérsékletet. Egyes berendezések a netről begyűjtik a várható időjárásra vonatkozó adatokat, és ezek alapján módosítják a fűtésszabályozást. Az okostermosztáthoz hasonlóan működnek a hálózati kapcsolattal rendelkező

légkondicionálók is. De vehetünk példának egy robotporszívót is, amely a mobiltelefonos alkalmazáson keresztül jelzi, ha végzett a napi takarítással, vagy üzenetet küldhet arról, hogy elakadt az ágy alá betolt doboz mögött, és ott kell majd keresni, mert nem tud önállóan előjönni. Ha az otthonunkból egy kicsit kilépünk, akkor az autónkban is felfedezhetjük a hálózati kapcsolódás lehetőségét. Egyre több új autó képes a fedélzeti számítógépén futó program frissítésére internetkapcsolaton keresztül.

Míg ötven-hatvan évvel ezelőtt néhány számítógépet próbáltak összekötni, addig napjainkban ötven-hatvan milliárd eszköz használja a világhálót. Ez azt jelenti, hogy ezekből az eszközökből már egy nagyságrenddel több van, mint ahány ember él a Földön.

Hálózattal kapcsolatos fogalmak

A hálózat kiterjedése

Először nézzük meg a hálózatok csoportosítását méret szerint! A legkisebb kiterjedésű hálózat a **személyi hálózat** (Personal Area Network, PAN). Itt néhány – legfeljebb tíz – méteren belüli eszközökről beszélünk. Ilyen hálózat lehet a számítógép és a vezeték nélküli egér vagy a billentyűzet között, vagy például a fülhallgatónk, a fitneszkarkötőnk, az okosmérlegünk és a telefonunk között. Itt leggyakrabban a vezeték nélküli Bluetooth vagy wifi biztosítja az összekapcsolódást.

▶ Optikai kábel

Helyi hálózatról (Local Area Network, LAN) akkor beszélünk, ha az egy épületen belüli, néhány tíz méter kiterjedésű. Ez a hálózat jellemzően a számítógépek, a munkaállomások, a szerverek összekapcsolására használható. Szinte minden internetelérással rendelkező családnál már üzemel egy helyi hálózat, amelyre a családtagok számítógépei, mobiltelefonjai kapcsolódhatnak. LAN működik az iskolában, a munkahelyeken. Ez lehetővé teszi a fájlok megosztását vagy a hálózati nyomtató közös használatát. A helyi hálózat szolgáltatásain keresztül lehet a mobiltelefonon tárolt képeket vagy videókat egy okostévén megnézni. A hálózati kapcsolat lehet vezetékes (Ethernet) vagy vezeték nélküli (wifi). Ezekről a technológiákról később még olvashatunk.

Napjainkban a helyi hálózatok nem elszigetelten működnek, hanem más helyi hálózatok is elérhetők belőlük. Egy több városban vagy országban működő vállalat esetén az egyes telephelyeken helyi hálózat üzemel, de a telephelyek összeköttetéséhez már **nagy kiterjedésű hálózat** (Wide Area Network, WAN) szükséges. Ezek a hálózatok lehetnek zártak, mint például egy banki rendszer, de kapcsolódhatnak az internethez is.

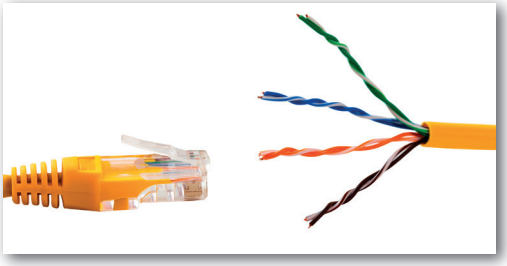
Amennyiben az informatikai eszközeink elhagyják a Földet, **bolygóközi hálózatról** beszélhetünk. A Naprendszerünk bolygóinak megfigyelésére küldött űrhajók, műholdak és a földi űrközpontok alkotnak ilyen hálózatot. Ebben az esetben már komolyan számolni kell az adattovábbítás időigényével, ugyanis a Föld és Mars közötti távolság megtételére a fénynek akár tíz percnél is több időre lehet szüksége.

Átviteli közegek

A hálózaton az adatok továbbításának módja a fizikai közegtől függően többféle lehet. Megkülönböztetjük a vezetékes és a vezeték nélküli adatátvitelt.

A **vezetékes adatátvitel** napjainkban általában csavart érpár, koaxiális kábel vagy üvegszálas kábel segítségével történik.

A **csavart érpáras kábellel** legtöbbször a helyi hálózatok kiépítésénél találkozunk. Ilyen kábelt használnak az Ethernet-hálózatokhoz. Leggyakrabban UTP- (Unshielded Twisted Pair: árnyékolatlan csavart érpár) kábelnek nevezik. Műszaki paraméterei miatt ez a kábel száz méternél nagyobb távolságra nem tud megbízhatóan adatot továbbítani.



► UTP-kábel

A nagy kiterjedésű hálózatok kialakításánál gyakran alkalmazzák a **koaxiális kábeleket,** amelyekkel a jelet kilométerekre is könnyen továbbíthatják. Korábban csak ilyen kábeleket használtak a kábeltelevíziós adások továbbítására, és ezeken a kábeleken keresztül is megoldható a számítógépes adatforgalom lebonyolítása.

► Koaxiális kábel

Napjainkban a legnagyobb távolságokat **optikai kábelekkel** hidalják át. Ilyen kábeleket fektetnek le az óceánok mélyén a kontinensek közti adatforgalom biztosítására is. A nagy áthidalható távolság mellett előnyük még más kábelekkel szemben, hogy az elérhető adatátviteli sebesség itt a legnagyobb. Az optikai kábelekben – a csavart érpáras és a koaxiális kábellel szemben – nem elektromos, hanem fényjeleket továbbítanak. Ez a kábel a legérzékenyebb a nem megfelelő hajlításra. Ha túl kis sugár mentén hajlítjuk meg a kábelt, az üvegszál eltörhet benne, így adattovábbításra alkalmatlanná válik.

Vezeték nélküli adattovábbításra többféle lehetőségünk van. A használt módszerek közül a leglassabb az **infravörös fényt** használó adatátvitel. Ilyen elven működik a televízió távirányítója, ahol az átviendő adat például a hangerő növelése vagy a nézni kívánt csatorna száma. Ennél az átvitelnél a minimális mennyiségű adat miatt a sebesség nem kritikus. Az infravörös jeleket nem zavarják a közelben működő elektromos berendezések, de fizikai takarással megszakítható a jeltovábbítás.

Lézeres adattovábbítás esetén közvetlen rálátást kell biztosítani a két eszköz között. A precízen irányított fény miatt a legkevésbé lehallgatható a jel. Kis távolság esetén nagy adatátviteli sebesség érhető el. Szabadtéren a köd akadályozhatja a működését, de hazánkban a javasolt telepítési előírások esetén évente néhány órányi kieséssel kell csak számolni.

Mikrohullámú adótornyok segítségével akár száz kilométer távolságra is lehet nagy sebességgel adatot továbbítani. A jel könnyen lehallgatható, ezért az adatok védelme érdekében mindenképpen titkosításra van szükség.

Az igazán helyfüggetlen megoldás a **műholdas adattovábbítás.** Ekkor az Egyenlítő felett 36 ezer kilométer távolságban keringő műholdról érkező jelet kell venni, és oda kell a jelet küldeni. A távolság miatt itt már komoly késleltetéssel kell számolni, amely másodpercekben mérhető.

A **Bluetooth** 2,4 GHz-es frekvencián működik, kicsi az energiafogyasztása, jellemzően legfeljebb tízméteres körzetben használható. Egy eszközhöz – például mobiltelefonhoz – egy időben legfeljebb hét aktív eszköz csatlakozhat. Ezek miatt a tulajdonságai miatt a hordozható eszközökben használják elterjedten, például fejhallgatóban, okosórában, pulzusmérőben vagy a kerékpárra szerelhető fedélzeti számítógépben.

A mikrohullámú adattovábbítás a **wifiszabványoknak** köszönhetően az otthonokban és az irodákban is megjelent. A szabványnak több változata van, ezeket a technikai fejlődéshez igazodva folyamatosan fejlesztik. Az adattovábbítás 2,4 GHz-en és 5 GHz-en történik. A tapasztalatok szerint 2,4 GHz-en lassabban, de nagyobb távolságra lehet továbbítani a jeleket, mint az 5 GHz-es frekvencián. Az adattovábbítás sebességét elsősorban a használt szabvány határozza meg, de a környezet, a falak száma, anyaga és vastagsága is komoly hatással van rá.

Sebesség

A hálózat sebességét több értékkel határozhatjuk meg. Az egyik a **késleltetés,** vagyis hogy az egyik berendezéstől a másikig mennyi idő alatt érkezik meg a jel. Ezzel a mindennapokban akkor találkozunk, amikor az interneten kommunikálva fontos számunkra a közel egyidejűség. Ahogy azt korábban olvashattuk, ez a Föld–Mars-viszonylatban akár több mint tíz perc is lehet. Ekkora távolságból már nem lehet közvetlenül irányítani egy leszállóegységet, mivel az egységről érkező jelek sok esetben csak azt mutatják a földi központ munkatársainak, hogy mi történt tíz perccel ezelőtt a Marson. A műholdas kapcsolat esetén a késleltetés több másodperc is lehet. A legjobb értékeket általában a vezetékes hálózatokon lehet elérni, azon belül is az optikai kábeles kapcsolat értéke a legnagyobb. Egy online akciójáték esetén a késleltetés értéke alapvetően határozza meg a játékélményt. Ebben az esetben csak a néhány ezredmásodperc az elfogadható érték a profiknak.

A hálózati sebesség másik fontos mérőszáma a **sávszélesség.** Ez az érték azt határozza meg, hogy egységnyi idő alatt mennyi adatot lehet továbbítani a hálózaton. Mivel az internetszolgáltatók, a távközlési vállalatok jellemzően nem szimmetrikus kapcsolatot biztosítanak, ezért megkülönböztetjük a feltöltési és a letöltési sávszélességet. A **feltöltési sávszélesség** azt mutatja meg, hogy a mi eszközünktől a másik eszközig (jellemzően a szerverig) másodpercenként mennyi adatot lehet eljuttatni. A letöltési sávszélesség pedig ennek a fordítottja, azaz a szervertől a mi eszközünkig mennyi adat tud egy másodperc alatt megérkezni. Az internetszolgáltatóknál általában a letöltési sávszélesség szokott nagyobb lenni, az internet-előfizetésnél ezt az értéket hangsúlyozzák.

Vegyünk egy konkrét példát! A szolgálató azt hirdeti, hogy a kínált csomag 500/40 Mbps (megabit per second: megabit másodpercenként). Ezt röviden 500-as netnek szokták mondani, függetlenül attól, hogy mekkora a kisebb érték. Mit is jelent ez? A letöltési sávszélesség maximuma 500 Mbps, a feltöltésé 40 Mbps. Ez az érték nem garantált, csak annyit jelent, hogy ideális esetben akár ennyivel is működhet a kapcsolatunk. A szolgáltatók a szerződéshez megadják a garantált sávszélességet is, amely a számunkra mindenképpen biztosított értéket jelenti. Nézzük ezt a fenti példának megfelelően! Az 500/40 Mbps mellett garantált sávszélességként megadják nekünk a 100/10 Mbps értéket. Ha ezt a sebességet nem bizto-

sítják, akkor jelezhetjük a szolgáltatással kapcsolatos minőségi kifogásunkat. A tapasztalat azt mutatja, hogy a szolgáltatók igyekeznek a maximálishoz közeli értéket biztosítani, mert így van esélyük az ügyfeleket megtartani, illetve újabbakat szerezni.

Hálózati berendezések

A hálózat egyes eszközei között az adattovábbításhoz több berendezésre van szükség. A vezetékes helyi hálózat esetén az egyes eszközök összekötéséhez **kapcsoló** (switch) szükséges. Ha például otthon a szobánkban egy UTP-kábelen jut be a hálózati kapcsolat, de több eszközt is szeretnénk csatlakoztatni, akkor megoldás lehet egy kapcsoló beszerzése, amelybe bedugjuk a meglévő UTP-kábelt, és minden eszközt újabb UTP-kábellel beköthetünk. Otthoni használatra vehetünk négy, nyolc vagy tizenhat csatlakozóval (porttal) rendelkező változatot.

► Kapcsoló (switch)

Vezeték nélküli kapcsolathoz szükségünk van egy **hozzáférési pontra** (Access Point, AP), amely a vezetéken érkező jelet a wifi-szabványnak megfelelő mikrohullámú jelekké alakítja, és lehetővé teszi az eszközöknek a hálózathoz való vezeték nélküli kapcsolódását.

A helyi hálózatok összekötése vagy a helyi hálózat internethez kapcsolása egy **forgalomirányítón** (routeren) keresztül valósítható meg.

Az otthoni internetkapcsolatunk biztosításához szükség van egy olyan berendezésre, amely a szolgáltató által biztosított kábelből jövő jelet képes megfelelően átalakítani. Itt elsősorban a koaxiális vagy optikai kábelből érkező jel átalakításáról van szó. Ezt az eszközt **modemnek** nevezik.

Milyen berendezésekre lesz szükségünk, ha szeretnénk otthon internetezni? Először is a szolgáltató optikai kábelt húz a lakásunkig. Ekkor kell egy *modem,* hogy az optikai jelet átalakítsuk. Ezután egy *routerre* lesz szükség, hogy a lakásunk helyi hálózatát az internethez kapcsolhassuk, a hálózatunkból érkező kérések eljuthassanak a szerverekhez, és onnan a válaszok beérkezzenek hozzánk. Ha több számítógépet szeretnénk vezetékesen a netre kapcsolni, akkor szükségünk lesz a helyi hálózatunkon belül egy *switchre* is. Ha a mobiltelefonunkról is használni akarjuk az otthoni internetelérést, akkor a vezeték nélküli kapcsolat biztosításához egy *AP* is szükséges. Tehát kell egy *modem,* egy *router,* egy *switch* és egy *AP*. A szolgáltató az internet bekötésével egy berendezést is szokott adni, ebben végződteti a saját kábelét. Ezen a készüléken több csatlakozási pont van, amelyhez vezetékesen kapcsolhatjuk eszközeinket, és jellemzően vezeték nélkül is kapcsolódhatunk hozzá. A köznyelvben **wifirouternek** nevezzük ezt a berendezést, amely a fent felsorolt négy eszközt egyben tartalmazza.


► Otthoni wifirouter

Felhőszolgáltatások

Felhőszolgáltatásoknak hívjuk azokat az informatikai megoldásokat, ahol egy üzleti vállalkozó a saját hardvereszközein üzemelteti a szolgáltatásokat, és a felhasználó elől az üzemeltetés részletei rejtve maradnak. Ilyenkor a felhasználó nem tudja, és nem is fontos tudnia, hogy fizikailag mely számítógépeken biztosítják számára a szolgáltatást. A felhasználó annyit tapasztal, hogy az internethez kapcsolódva bárhol is van, mindig azonos módon éri el a szolgáltatásokat. A **helyfüggetlenség** mellett fontos, hogy **méretezhetőség,** más néven **skálázhatóság** jellemzi, azaz az igényeknek megfelelően képes növekedni. Amennyiben egy vállalkozás ilyen szolgáltatást vesz igénybe, és megnő a vállalkozás forgalma, akkor a szerverközpontokban üzemelő szolgáltatás több vagy nagyobb teljesítményű szerverekre tud automatizált formában költözni. Ennek természetesen növekvő költségei lesznek, de nagyobb forgalomból várhatóan nagyobb bevétel is következik. Mivel a szerverek üzemeltetését erre szakosodott szervezet végzi, a szolgáltatás **rendelkezésre állása** is magasabb, kevesebb rendszerhibából adódó időkieséssel kell számolni. Ez átlagosan éves szinten legfeljebb néhány perc. Egy szerverközpontban a feladatok a nagyszámú gép között jól eloszthatók, így egy cég igényeinek megfelelő szolgáltatás sokkal költséghatékonyabb, mint ha azt saját eszközökkel, saját alkalmazásban lévő rendszergazdával kellene megoldania.

A felhőszolgáltatásoknak az egyik nagy előnyük, hogy több lehetőséggel támogatják a csapatmunkát. Sokszor az alapszolgáltatásokat ingyenesen elérhetővé teszik a szolgáltatók, csak a nagyobb tárterületért, speciális funkciók eléréséért, esetleg a reklámmentességért kell havidíjat fizetni.

Nézzük a teljesség igénye nélkül, hogy egy projekthez a felhőszolgáltatásokból mit vehetünk igénybe. Legyen az első a kommunikáció!

Lehetőségünk van **levelezőszolgáltatást** igénybe venni. Sokak által használt a Gmail, az Outlook, a Microsoft365, a ProtonMail vagy a Yahoo Mail levelezőszolgáltatás.

A levelezés mellett a kapcsolattartás rövid, azonnali szöveges üzenetekben is zajlik, ez a **csevegés** (angolul chat). Gyakran használt csevegőszolgáltatás a Messenger, a Viber, a Skype és a WhatsApp. Ha hangalapon szeretnénk kommunikálni, akkor azt általában a csevegőprogramokkal is megtehetjük.

Videóhívásokra is lehetőséget szoktak biztosítani a csevegőprogramok, de vannak olyan szolgáltatások, amelyek kifejezetten ebben erősek. Ilyen például a Microsoft Teams, a Webex Meetings, a Zoom vagy a Google Meet.

► Videókonferencia

Ha állományokat szeretnénk tárolni, másokkal megosztani, közösen használni, akkor a felhőalapú tárhelyszolgáltatásokat vehetjük igénybe, például a Microsoft OneDrive-ot, a Google Drive-ot vagy a Dropboxot. Ezeknél lehetőségünk van az automatikus szinkronizálás beállítására. Ha beállítjuk, akkor a telefonunkon készített fényképek automatikusan a felhőalapú tárterületünkre másolódnak, így nem kell külön gondoskodnunk a biztonsági másolatukról. Ilyenkor beállíthatjuk, hogy a mobilhálózati adatforgalom esetén is engedélyezett legyen

a szinkronizálás, ami külön költségeket jelenthet, vagy hogy csak wifikapcsolat esetén induljon a másolás. Ehhez hasonlóan a számítógépünkön tárolt állományokhoz is beállíthatjuk a szinkronizálást, amivel a biztonsági mentésünk lesz biztosított. Azon túl, hogy a magunk számára eltárolhatunk állományokat, lehetőségünk van őket másokkal is megosztani. A megosztásnál meghatározhatjuk, hogy az állományokat mindenki elérheti-e, vagy csak az általunk engedélyezett személyek. Beállíthatjuk, hogy csak megtekinteni tudják, vagy lehetőségük legyen módosítani is őket. Van olyan szolgáltatás, ahol még időkorlátot is be lehet állítani, például három napig elérhető a linken, utána már nem lesz megosztott. Ha egy projektfeladat kapcsán a mobiltelefonunkkal készítünk egy videót vagy fényképeket, akkor a telefonunkról a felhőtárhelyre másolhatjuk, majd a csapatunk többi tagjával megoszthatjuk őket. Ha egy megosztott mappába újabb állományokat teszünk, akkor a többiek számára azonnal elérhetővé válnak.

Ha **dokumentumokat** szeretnénk **szerkeszteni** a többiekkel egy időben, akkor használhatjuk például a Google Dokumentumok vagy a Microsoft365 szolgáltatást.

A felsorolt felhőszolgáltatásokban közös, hogy igénybevételük **eszközfüggetlen,** azaz laptopon, táblagépen, okostelefonon egyaránt elérhetők, sok esetben még alkalmazást sem kell telepíteni, elegendő egy böngészőből csatlakozni a szolgáltatás weboldalához. Itt egy felhasználói azonosítót és jelszót kell általában megadni. A biztonság növelhető az úgynevezett kétfaktoros hitelesítéssel, amikor a bejelentkezéshez nem elegendő e két adat, hanem egy fizikai eszközre is szükség van. A leggyakoribb megoldás, hogy a bejelentkezéskor kapunk a mobiltelefonunkra egy egyszer használatos kódot, melyet a bejelentkezési felületen meg kell adnunk, vagy csak a mobiltelefonunkon nyugtázni kell a bejelentkezési kísérletet. Erre azért van szükség, mert egy felhasználónév és jelszó párost megszerezhetnek például egy kémprogram segítségével, de a kétfaktoros azonosítás révén nem tudnak a nevünkben bejelentkezni, ha a telefonunk mindeközben nálunk van.

Kérdések, feladatok

1. Mi a különbség az UTP-, az STP- és az FTP-kábelek között?
2. Milyen eszközöket lehet használni egy laptoppal Bluetooth-kapcsolaton keresztül?
3. Nézzünk utána, melyik szolgáltatóktól lehet vezetékes internetelérést rendelni a lakóhelyünkön! Milyen minimális és maximális sávszélességeket kínálnak? Mennyi a havidíjuk az egyes szolgáltatásoknak?
4. Mit lehet tenni akkor, ha az egész lakásban szeretnénk vezeték nélkül netezni, de a lakás egyik végén elhelyezett wifirouter jele nem érhető el a lakás másik végén, és a routert nem lehet áthelyezni a szolgáltató kábele miatt?
5. Milyen felhőszolgáltatásokat érdemes használni egy 3–4 fős csapatban megoldandó projekt kapcsán? A különböző projektek a használandó eszközökben eltérhetnek, így érdemes többféle projekt esetében is végiggondolni a választ.

A digitáliskultúra-tankönyvek digitális változatai: az okostankönyvek

A Nemzeti Köznevelési Portálon (www.nkp.hu) található okostankönyvek a digitális kiegészítő tartalmak révén segítenek az általános ismeretek megszerzésében és elmélyítésében. Rendszeres használatukkal még élvezetesebb és eredményesebb lehet a digitális kultúra tanulása.

Az okostankönyv olyan digitális tankönyv, amely egyesíti a korszerű technológiákkal létrehozott weblapok és webes szolgáltatások funkcióit. Olvasásához és használatához mindössze egy böngészőprogram szükséges. A HTML5 technológiának köszönhetően a tankönyv teljes szöveges tartalma kereshető, mivel a böngészők „belelátnak” a leckékbe.

A kiadásért felel: Brassói Sándor mb. elnök
Raktári szám: OH-DIG11TA
Tankönyvkiadási osztályvezető: Horváth Zoltán Ákos
Műszaki szerkesztő: Görög Istvánné
Nyomdai előkészítés: Korda Ágnes
Terjedelem: 21,45 (A/5) ív, tömeg: 510 gramm
1. kiadás, 2022

Gyártás: Könyvtárellátó Nonprofit Kft.

Nyomtatta és kötötte:
Felelős vezető:
A nyomdai megrendelés
törzsszáma: