OKTATÁSI HIVATAL

NAT 2020

10

Digitális kultúra

tankönyv

DIGITÁLIS KULTÚRA 10.

OKTATÁSI HIVATAL

A kiadvány 2026. 08. 31-ig tankönyvi engedélyt kapott a TKV/97-7/2021 számú határozattal. A tankönyv megfelel a Kormány 5/2020 (I.31.) Korm. rendelete a Nemzeti alaptanterv kiadásáról, bevezetéséről és alkalmazásáról szóló 110/2012. (VI.4.) Korm. rendelet módosításáról megnevezésű jogszabály alapján készült Kerettanterv a középiskola 10. évfolyama számára megnevezésű kerettanterv Digitális kultúra tantárgy előírásainak.

A tankönyvvé nyilvánítási eljárásban közreműködő szakértő: Györgyi Tamás

Tananyagfejlesztők: Abonyi-Tóth Andor, Farkas Csaba, Jeneiné Horváth Kinga, Reményi Zoltán, Tóth Tamás, Varga Péter
Kerettantervi szakértő: Siegler Gábor
Lektor: Farkasfalvy Judit
Szerkesztő: Pintér Gergely
Fedélterv: Slezák Ilona
Fotók: Shutterstock (Wachiwit, Twin Design, Mia2you)

ISBN 978-615-6256-39-3

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

A kiadásért felel: Brassói Sándor mb. elnök
Raktári szám: OH-DIG10TA
Tankönyvkiadási osztályvezető: Horváth Zoltán Ákos
Műszaki szerkesztő: Görög Istvánné
Nyomdai előkészítés: Korda Ágnes
Terjedelem: 11,8 (A/5) ív, tömeg: 420 gramm

1. kiadás, 2021

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.

Gyártás: Könyvtárellátó Nonprofit Kft.

Nyomtatta és kötötte:
Felelős vezető:
A nyomdai megrendelés törzsszáma:

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE

Tartalom

Előszó

Kedves Diákok!

Alig van olyan terület, kommunikáció, ügyintézés a környezetünkben, amely ne digitálisan történne. Láthatóan, vagy éppen elrejtve a szemünk elől, egyszerre adatokat szolgáltatunk, ugyanakkor nagy mennyiségű adat feldolgozásának eredményeit használjuk fel. Teljesen megváltozott az információszerzés módja, sebessége és felhasználási köre. Menetrendet már nem könyvben, sőt nem is offline elektronikus úton, hanem online, a világhálón keresünk. Az útvonaltervező alkalmazások a két pont közötti közlekedés feltételeit már régóta megadják, de a hálózatos alkalmazások a közlekedés pillanatnyi helyzetét is figyelembe veszik. Értenünk kell, hogy az ehhez szükséges forgalmi adatokat honnan gyűjtik a térinformatikai rendszerek, milyen feldolgozási elv szolgáltatja ezeket.

► BIG DATA: Adatfelhőben élünk…

A digitális kultúra tantárgy különböző témáinak középpontjában az adat, az adatfeldolgozás és az adattovábbítás áll. Az eszközök hozzáértő alkalmazása mellett fontos, hogy felkészüljünk új problémák digitális eszközökkel történő megoldására. Megismerjük az informatika tudományterületét addig a szintig, hogy értsük a környezetünkben végbemenő folyamatokat, és ha szükséges, célszerűen be is tudjunk avatkozni. Fontos szerepet kap a megváltozott kultúránk, az új eszközök felelősségteljes használata.

A tankönyv első részében folytatjuk az adatkezelés módszereinek megismerését, bővítjük a táblázatkezelési ismereteinket, nagyobb adatmennyiségekkel dolgozunk. Lényeges elvárás lesz az eredmények különböző szempontú és technikájú megjelenítése.

A második részben az online kommunikáció módszereit, az információs hálózatok alkalmazásának eszközeit, használatát tárgyaljuk. Az előzetes ismereteinket rendszerezzük, bővítjük. Folytatjuk az előző évben már elkezdett *Publikálás a világhálón* című témát. A HTML-formátumú dokumentumok szerkezeti elemeinek, valamint a CSS-használat alapelveinek megismerése után a webes tartalomkezelő rendszerekkel és az ezekben való publikálással foglalkozunk.

A harmadik részben az informatikai eszközökkel és módszerekkel történő problémamegoldással kapcsolatos ismereteinket bővítjük. A strukturált programozás alapfogalmait, módszereit a problémák megoldásához használjuk. Ehhez a Pythont használjuk a gyakorlatban, de fontos hangsúlyoznunk, hogy a választott nyelv csupán egy eszköz, amelyet most vagy később más elterjedt nyelv(ek) válthatnak fel.

A táblázatkezelés és a programozás után az adatkezelés alapfogalmaival, az adatbázis-kezelés eszközeivel kezdünk el foglalkozni, amit a következő évben fogunk folytatni.

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

A tankönyv szerves részét képezik az elérhető elektronikus anyagok, fájlok, amelyek a https://www.tankonyvkatalogus.hu/site/kiadvany/OH-DIG10TA oldalról tölthetők le.

Jó munkát, a tankönyv eredményes használatát kívánják a szerzők!

Táblázatkezelés

Problémamegoldás táblázatkezelővel

Ebben a fejezetben arra mutatunk be példákat, hogyan használhatjuk a táblázatkezelő programot más tantárgyakban, pl. földrajz-, matematika-, fizika- vagy nyelvtanórán.

Az Európai Unió országai

Első példánkban az Európai Unió adatait elemezzük a Központi Statisztikai Hivatal adatai alapján. A 2019-es adatokat a könyv weblapjáról elérhető *eu.xlsx* fájl tartalmazza. A táblázat oszlopai rendre a tagország nevét, az EU-ba való belépés évét, területét, népességét, GDP-jét, a születéskor várható élettartamot, a főváros nevét és a főváros lakóinak számát tartalmazza. A mértékegységeket a táblázat első sorában tüntettük fel.

	A	B	C	D	E	F	G	H	I	J
1	Ország	Belépés	Terület ezer km^2	Népesség millió fő	Néps fő/km^2	GDP mrd €	GDP egy főre (€)	Sz.v.ét. év	Főváros	Főv. nép. ezer fő
2	Ausztria	1995	83,88	8,88		398,68		81,80	Bécs	1 915,30
3	Belgium	alapító	30,53	11,50		473,09		81,70	Brüsszel	2 065,30
4	Bulgária	2007	110,37	6,98		60,68		75,00	Szófia	1 276,90
5	Ciprus	2004	9,25	0,88		21,94		82,90	Nicosia	269,50
6	Csehország	2004	78,87	10,67		220,20		79,10	Prága	1 298,80
7	Dánia	1973	42,92	5,81		310,94		81,00	Koppenhága	1 333,90

► Az Európai Unió tagországainak adatai a KSH alapján (Forrás: https://www.ksh.hu/stadat_eves_7)

Határozzuk meg az *E* oszlop celláiban, hogy mekkora az egyes országok népsűrűsége! A népsűrűséget a népesség és a terület hányadosa adja, ez például Ausztria esetén *D2/C2* lenne. Mivel a népesség millió főben, a terület pedig ezer km^2-ben van megadva, a hányadost még szorozni kell 1000-rel is, így a `=D2/C2*1000` képlet került az *E2*-es cellába. Hasonló módon a *G2*-es cellában az egy főre jutó GDP-t €-ban kifejezve az `=F2/D2*1000` képlettel kapjuk. (Érdemes megvizsgálnunk ezeket az értékeket: jelentős eltéréseket találunk az egyes tagországok között.)

Az *L2:N17* tartományban további kérdésekre keresünk választ. Például az *M2:M4* tartomány celláiban meghatározzuk az unió területét és népességét, ebből pedig kiszámoljuk a népsűrűségét:

```
M2: =SZUM(C2:C28)
M3: =SZUM(D2:D28)
M4: =M3/M2*1000
```

	L	M	N
1			
2	EU területe	4 131,93	ezer km^2
3	EU népessége	447,27	millió fő
4	EU átlagos népsűrűsége	108,25	fő/km^2
5			
6	Legnagyobb GDP	3 435,21	mrd €
7	GDP-je a legnagyobb	Németország	
8			
9	Magyarországnál nagyobb a…		
10	területe	10	
11	területe és népessége	8	
12	területe, népessége és GDP-je	8	
13			
14	1 főre jutó GDP		
15	teljes EU	31 139,07	€
16	alapítók	37 493,69	€
17	2000 után belépők	14 601,41	€

► További kérdések és a válaszok

Melyik ország GDP-je a legnagyobb az Unióban? Határozzuk meg először a legnagyobb GDP értékét! A legnagyobb GDP-t az *M6*-os cellában a `=MAX(F2:F28)` képlet adja. Azt, hogy ez melyik országhoz tartozik, meghatározhatjuk például az *INDEX…HOL.VAN* függvényekkel, így az *M7*-es cellában a következő képlet szerepel:

```
=INDEX(A2:A28;HOL.VAN(M6;F2:F28;0))
```

Vajon hol helyezkedik el Magyarország az unió 27 tagországa között? Hány nálunk „nagyobb" található? Pontosabban hány olyan ország van, melynek hazánkénál nagyobb a területe; nagyobb a területe és a népessége, illetve nagyobb a területe, a népessége és a nemzeti jövedelme is? A három kérdésre a válaszokat például a *DARABTELI* és *DARABHATÖBB* függvénnyel kapjuk az *M10*-es, *M11*-es és *M12*-es cellákban:

```
=DARABTELI(C2:C28;">"&C19)
=DARABHATÖBB(C2:C28;">"&C19;D2:D28;">"&D19)
=DARABHATÖBB(C2:C28;”>”&C19;D2:D28;”>”&D19;  F2:F28;”>”&F19)
```

Az Európai Unió 1 főre jutó GDP-jét – a népsűrűség számításához hasonlóan határozhatjuk meg az *M15*-ös cellában:

```
=SZUM(F2:F28)/SZUM(D2:D28)*1000.
```

Kicsit összetettebb a feladat, ha szeretnénk meghatározni az egy főre jutó GDP-t az *alapító* országok esetén, illetve a *2000 után belépő* országokban külön-külön is. Ezúttal feltételes összegzést kell végeznünk, amit például a *SZUMHATÖBB* függvény segítségével készíthetünk, így a két képlet az *M16*-os és *M17*-es cellákban:

```
=SZUMHATÖBB(F2:F28;B2:B28;"alapító")/
                SZUMHATÖBB(D2:D28;B2:B28;"alapító")*1000
=SZUMHATÖBB(F2:F28;B2:B28;">"&2000)/
                SZUMHATÖBB(D2:D28;B2:B28;">"&2000)*1000
```

Feladatok

1. Milyen következtetéseket vonhatunk le a kapott adatokból? Hazánk az EU többi országához képest nagy vagy kicsi? Jóval fejlettebbek-e az alapítók, mint a 2000 után belépők?
2. Határozzuk meg az *M19*-es cellában, hogy melyik ország fővárosa a legnépesebb!
3. Képlet alkalmazásával tegyünk a *K* oszlopba egy „+" jelet, ha az adott országban az egy főre jutó GDP az uniós átlagnál nagyobb, egyébként pedig egy „-" jelet!
4. Határozzuk meg képlettel az *M21*-es cellában, hogy hány olyan ország van, amelyben az egy főre jutó GDP nagyobb az uniós átlagnál! Hány ember él ezekben az országokban? A választ az *M22*-es cellában jelenítsük meg!
5. Vajon van-e kapcsolat az egy főre jutó GDP és a születéskor várható élettartam között? Hogyan tudnánk erre a kérdésre a táblázatkezelő eszközeinek felhasználásával választ adni?

Egyenletek megoldása

Másodfokú egyenlet megoldása

Mint matematikaórán tanultuk, az $ax^2 + bx + c = 0$ ($a \neq 0$) másodfokú egyenlet megoldása a $D = b^2 - 4ac$ diszkriminánstól függ. Ha $D < 0$, akkor nincs valós megoldás, ha $D = 0$, akkor egy valós megoldás van, ha pedig D > 0, akkor kettő. Ez utóbbi esetben a két valós megoldást az $x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ megoldóképlet adja. Először ezt a megoldóképletet használjuk fel az egyenlet megoldására a táblázatkezelő program segítségével.

Hozzuk létre a táblázat munkalapján az alábbi ábrán látható feliratokat az 1. sorban, továbbá az *A* és *D* oszlopokban!

Ha feltételezzük, hogy a felhasználó olyan együtthatókat választott, amelyek két valós megoldást adnak, akkor például az *E2* és *E3* cellákba kerülő képletek:

E2: `=B3*B3-4*B2*B4,` *E3:* `=(-B3+GYÖK(E2))/(2*B2)`

Hogyan tudnánk a képleteket úgy továbbfejleszteni, hogy azok akkor is helyesen működjenek, ha a diszkrimináns negatív? Mivel ebben az esetben a *GYÖK()* függvény hibát jelez, lehetőségünk van a **HAHIBA***(kif1, kif2)* hibakezelő függvény alkalmazására. Ha a *kif1* kifejezés hibátlanul működik, akkor ez a függvény annak eredményét írja a cellába, egyébként pedig a *kif2* értékét. Így az *E2*-es cellába kerülő képlet:

```
=HAHIBA((-B3+GYÖK(E2))/(2*B2);"Nincs valós megoldás!")
```

	A	B	C	D	E	F
1	**Az $ax^2+bx+c=0$ alakú egyenlet megoldása**					
2	a=	**3**		D=	**76**	
3	b=	**4**		x_1=	**0,7863**	
4	c=	**-5**		x_2=	**-2,1196**	

	A	B	C	D	E	F
1	**Az $ax^2+bx+c=0$ alakú egyenlet megoldása**					
2	a=	**3**		D=	**-44**	
3	b=	**4**		x_1=	**Nincs valós megoldás!**	
4	c=	**5**		x_2=	**Nincs valós megoldás!**	

▶ Másodfokú egyenlet megoldása

Egyenletek grafikus megoldása

Matematikaórán elhangzott, hogy algebrai úton legfeljebb negyedfokú egyenletek oldhatók meg, ugyanakkor már a harmadfokú egyenletek megoldóképletének alkalmazása helyett is gyakran gyorsabbak a közelítő megoldások.

Ebben a példában a $-0{,}5x^3 + 0{,}8x^2 - 0{,}6x + 1 = 0$ harmadfokú egyenletet fogjuk megoldani. Értéktáblázattal határozzuk meg az $f(x) = -0{,}5x^3 + 0{,}8x^2 - 0{,}6x + 1$ függvény értékeit –1 és +2 között 0,2 tizedenként, majd ábrázoljuk az adatokat pontdiagramon!

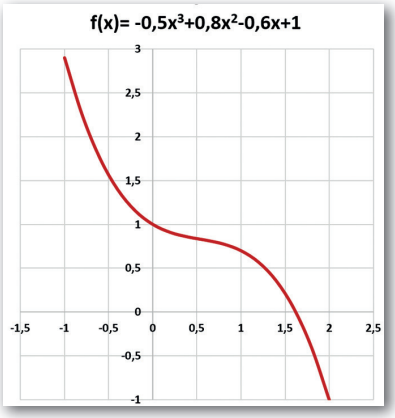
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	**x**	-1,00	-0,80	-0,60	-0,40	-0,20	0,00	0,20	0,40	0,60	0,80	1,00	1,20	1,40	1,60	1,80	2,00
2	**f(x)**	2,90	2,25	1,76	1,40	1,16	1,00	0,91	0,86	0,82	0,78	0,70	0,57	0,36	0,04	-0,40	-1,00

▶ Az $f(x) = -0{,}5x^3 + 0{,}8x^2 - 0{,}6x + 1$ függvény értékei –1 és 2 között 0,2-es lépésközzel, 2 tizedesjegy pontossággal

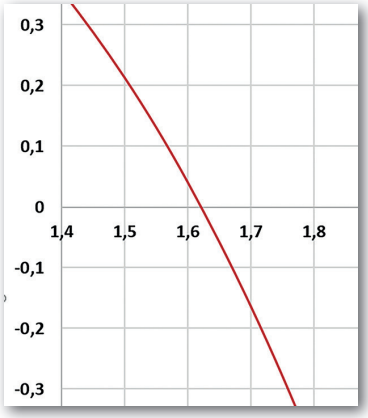
Az ábrán a *B2*-es cellába a képlet került:

```
=-0,5*B1*B1*B1+0,8*B1*B1-0,6*B1+1
```

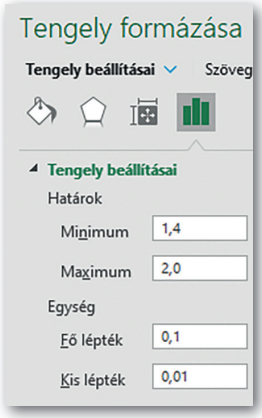
Az értéktábla alapján úgy tűnik, hogy az egyenlet gyöke 1,6 és 1,8 közé esik, erről a grafikonról is meggyőződhetünk. Ha a grafikonon csökkentjük a tengelyeken a lépésközt, akkor pontosabb eredményt is leolvashatunk: a gyök 1,62 körül van.



► A függvény grafikonja



► Csökkentettük a tengelyen a lépésközt (Microsoft Excel)

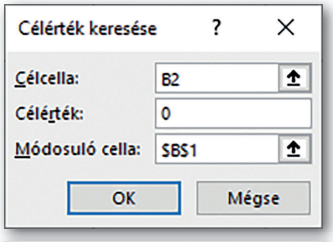


Egyenletek megoldása célértékkereséssel

A gyökre becslést kaphatunk a táblázatkezelő program célértékkeresés funkciójával is. Ezt például az *Adatok > Lehetőségelemzés > Célértékkeresés (illetve az Eszközök > Célértékkeresés)* menüponttal érhetjük el.

Használatához adjuk meg a vizsgált képletet tartalmazó cellát *(B2)*, a várt eredményt (0), illetve azt a cellát, amely a változót tartalmazza *(B1)*. A táblázatkezelő program a változót tartalmazó cella módszeres változtatásával megpróbálja elérni a kért eredményt.

Esetünkben az egyenlet gyöke x = 1,6209 körül van, ekkor a függvényérték már csak kb. 0,00003-del tér el zérustól.



	A	B
1	x	1,620886178
2	f(x)	0,000031458

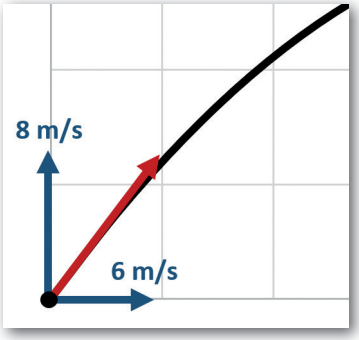
► Célértékkeresés

Feladatok

1. Bővítsük tovább a másodfokú egyenletet megoldó munkalap képleteit úgy, hogy az jelezze azt is, ha csak egyetlen megoldás van!
2. Ábrázoljuk közös koordinátarendszerben a következő függvényeket! Milyen kapcsolatot láthatunk az egyes függvények grafikonjai között?
   $f(x) = x^2$ $g(x) = -x^2$ $h(x) = (x-2)^2$ $i(x) = x^2 - 2$ $j(x) = -(x-2)^2 - 2$
3. Keressük meg célértékkereséssel a következő egyenletek gyökeit:
   a) $x^3 - 3x^2 + x - 3 = 0$
   b) $x^4 - x^3 - 7x^2 + x + 6 = 0$

Ferde hajítás

A ferde hajítás a fizika egyik érdekes és rendkívül gyakorlatias problémája: milyen pályán mozog egy elrúgott focilabda vagy egy kilőtt ágyúgolyó? Feladatunk az, hogy meghatározzuk a ferdén elhajított test *x* és *y* koordinátáit a mozgás során. Készítsük el a táblázatot a minta és a leírás alapján!



▶ Az *x* és *y* komponenst külön vizsgáljuk

Kezdetben legyen a test az origóban ($x = 0$ és $y = 0$), és hajítsuk el úgy, hogy kezdősebessége vízszintesen 6 m/s, függőlegesen pedig 8 m/s legyen! (Pitagorasz tételével könnyen ellenőrizhetjük, hogy a test kezdősebessége 10 m/s.) Ezeket az adatokat táblázatunk *A5:E6* tartományának celláiban találjuk.

A továbbiakban választunk egy viszonylag rövid Δt időközt (például $\Delta t = 0{,}1$ s), és felbontjuk a mozgás időtartamát Δt időegységekre. Minden időegység végén megadjuk a test helyének és sebességének vízszintes és függőleges komponensét az időegység elején kapott értékekből a fizika szabályainak alkalmazásával. Az időegység értékét táblázatunk *B2*-es cellájában rögzítettük.

Ha (egyelőre) eltekintünk a légellenállástól, a test vízszintes sebességkomponense nem változik, vagyis az első időegység végén megegyezik az időegység elején kapott értékkel, így a *B7*-es cella tartalma a

```
=B6
```

képlet lesz.

A test vízszintes irányú elmozdulásának meghatározása – gondolva arra, hogy később figyelembe szeretnénk venni a közegellenállást is – a következőképpen történik. Először meghatározzuk a test átlagsebességét az adott időegységre vonatkozóan: (*B6*+*B7*)/2. Majd ebből a $\Delta s = v_{átl}\Delta t$ képlet alkalmazásával kiszámoljuk a vízszintes irányú elmozdulást: (*B6*+*B7*)/2*\$*B*\$2, és ezt hozzáadjuk az időegység elején kapott értékhez. Így a *D7*-es cellába a következő képlet kerül:

```
=D6+(B6+B7)/2*$B$2
```

	A	B	C	D	E
1	g=	10	m/s²		
2	Δt=	0,1	s		
3	k=				
4					
5	**Idő**	**v_x**	**v_y**	**x**	**y**
6	0,0	6,00	8,00	0,00	0,00
7	0,1	6,00	7,00	0,60	0,75
8	0,2	6,00	6,00	1,20	1,40
9	0,3	6,00	5,00	1,80	1,95
10	0,4	6,00	4,00	2,40	2,40
11	0,5	6,00	3,00	3,00	2,75
12	0,6	6,00	2,00	3,60	3,00
13	0,7	6,00	1,00	4,20	3,15
14	0,8	6,00	0,00	4,80	3,20
15	0,9	6,00	-1,00	5,40	3,15
16	1,0	6,00	-2,00	6,00	3,00

▶ A ferde hajítást végző test pályájának és sebességének x és y irányú komponensei

A függőleges komponens vizsgálata (ha eltekintünk a légellenállástól) csak annyival bonyolultabb, hogy figyelembe kell venni a Föld tömegvonzását is.

A nehézségi gyorsulás *(g)* értékét a *B1*-es cellában rögzítjük (így később a számításokat más bolygókra is elvégezhetjük).

A test függőleges sebessége Δt időegység alatt $g\Delta t$-vel csökken, így a *C7*-es cellába a következő képlet került:

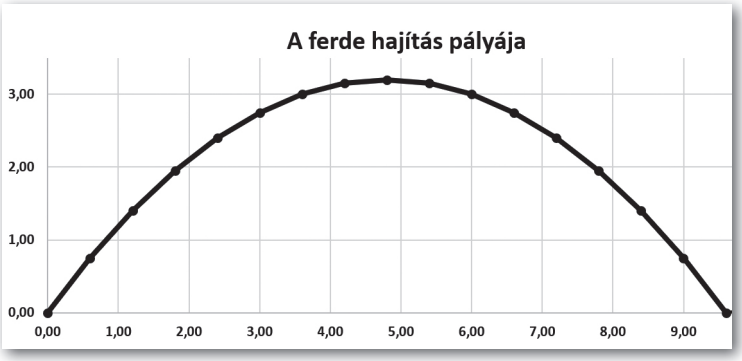
```
=C6-$B$1*$B$2
```

Végül a test *y* koordinátáját a korábban alkalmazott $\Delta s = v_{átl}\Delta t$ képlet alkalmazásával az *E7*-es cellában a

```
=E6+(C6+C7)/2*$B$2
```

képlettel kapjuk.

Az eredményt látványosabbá tehetjük, ha a mozgás során a koordinátákat pontdiagramon ábrázoljuk.



▶ A test koordinátáinak megjelenítése pontdiagramon

Feladatok

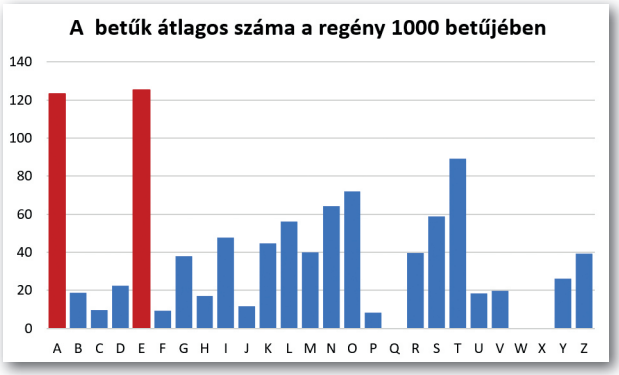
1. A képletek kialakítása során nem vettük figyelembe, hogy ha a test visszaér a felszínre ($x = 0$), akkor nem tud „lejjebb esni”. Hogyan kell módosítanunk ennek figyelembevételéhez a képleteket?
2. Hogyan mozogna a ferdén elhajított test a Holdon (g = 1,64 m/s^2), a Napon (g = 274 m/s^2), illetve egy olyan helyen, ahol nincs nagy tömegű test a közelben (g = 0 m/s^2)?
3. A légellenállás kis sebességek esetén a sebességgel egyenesen arányos, az arányossági tényező a test alakjától és felületétől függ. Példánkban ezt az arányossági tényezőt a *B3*-as cella tartalmazza, legyen értéke például 0,4. Hogyan változik a test pályája, ha a légellenállást is figyelembe vesszük?

Betűk gyakorisága

A betűk gyakorisága a magyar nyelvben

Először meghatározzuk az egyes betűk gyakoriságát egy magyar nyelvű szövegben. Töltsük le a könyv weboldaláról a *legyjo.txt* szöveges állományt, amely Móricz Zsigmond Légy jó mindhalálig! című regényének ékezetmentes változatát tartalmazza szóközök, számok és írásjelek nélkül.

Másoljuk be az *A2*-es cellától kezdve a szöveget a táblázatkezelő munkalapjára! A többi feliratot a minta szerint készítsük el! (A fájlban a 405 589 karakter mindegyike külön sorban szerepel.)



► A magyar nyelvben a két leggyakoribb betű az A és az E

	A	B	C	D	E	F	G
1	nyílt	titk		betű	db	k	kulcs
2	M	N		A	49960	123	T
3	O	C		B	7631	19	A
4	R	U		C	3854	10	B
5	I	P		D	9072	22	L
6	C	B		E	50786	125	Z
7	Z	J		F	3789	9	K
8	Z	J		G	15357	38	E
9	S	V		H	6923	17	O
10	I	P		I	19363	48	P
11	G	E		J	4736	12	R
12	M	N		K	18173	45	G
13	O	C		L	22754	56	M
14	N	W		M	16213	40	N
15	D	L		N	26123	64	W
16	L	M		O	29220	72	C
17	E	Z		P	3393	8	I
18	G	E		Q	7	0	Q
19	Y	F		R	16107	40	U
20	J	R		S	23872	59	V
21	O	C		T	36219	89	S
22	M	N		U	7458	18	X
23	I	P		V	8018	20	H
24	N	W		W	4	0	D
25	D	L		X	7	0	Y
26	H	O		Y	10620	26	F
27	A	T		Z	15930	39	J

► A betűk száma

Vegyük fel az ábécé betűit ékezetek nélkül a *D2:D27* tartományban, majd határozzuk meg azok gyakoriságát! A *DARABTELI* függvényt alkalmazva az *E2*-es cellában szereplő képlet: `=DARABTELI($A$1:$A$405590;D2)`.

A mintát könnyebb összehasonlítani egy másik szöveggel, ha a betűk darabszáma helyett azt adjuk meg, hogy az adott betű átlagosan hányszor fordul elő a szöveg 1000 karakterében. Például az *F2*-es cellában: `=E2/SZUM($E$2:$E$27)*1000`.

Az eredményt szemléltessük oszlopdiagramon!

Titkosítás más betűk behelyettesítésével

A titkosírások egyik régi technikája, amikor az ábécé egyes betűit az ábécé más betűivel helyettesítik. Egy ilyen titkosításhoz tartozó kódtáblát látunk az ábrán a *D2:D27;G2:G27* nem összefüggő tartományban.

A kódtábla alkalmazása a *B* oszlopban történhet például az *INDEX...HOL.VAN* függvénypárossal, így a *B2*-es cellába a következő másolható képlet írható:

`=INDEX($G$2:$G$27; HOL.VAN(A2;$D$2:$D$27;0))`

A titkosított szöveget a *B* oszlopból a vágólapon át könnyen áthelyezhetjük egy editorba (pl. a Notepad++), ahol a *Csere* funkcióval a sortörések eltávolíthatók.

A betűhelyettesítéssel kapott titkos szöveg feltörése

Vajon mennyire megbízható ez az eljárás? Matematikailag könnyen látható, hogy egy 26 betűs ábécé esetén az *A* betű 26-féleképpen helyettesíthető, a *B* 25-féleképpen, és így tovább, vagyis a lehetséges kódtáblák száma:

$26! \cong 400\,000\,000\,000\,000\,000\,000\,000\,000.$

Valamennyi eset kipróbálása a középkorban lehetetlen volt. Azonban már a 11. században felismerték, hogy az így titkosított szöveg a betűgyakoriság vizsgálatával megfejthető.

Töltsük be a könyv weblapjáról letölthető *titkos.txt* fájlt, és tartalmát illesszük a munkalap *A* oszlopába! Az előző példának megfelelően határozzuk meg, hogy az egyes betűk átlagosan hányszor fordulnak elő a szöveg 1000 karakterében! Az ábrán látható, hogy a *G* oszlopba felvettük az előző mintánál kapott megfelelő adatokat is.

Hasonlítsuk össze a mostani szövegben, illetve az előző példában végzett számításokat, vagyis az *F* és *G* oszlopok adatait! Az előző példa oszlopdiagramjáról leolvasható, hogy egy magyar nyelvű szövegben a leggyakoribb betű az *A* és az *E*. Így kialakulhat az a sejtésünk, hogy a titkos szövegben az *A* és *E* betű megfelelője az *O* és az *S* betű lehet. Vezessük át ezt a *B* oszlopba!

Tovább vizsgálva a szöveget, feltűnhet, hogy a vélhetőleg *E* betűvel kezdődő szövegrészek esetén a titkos szövegben az első három betű gyakran az *SHQ* betűhármas. Mivel a magyarban gyakori az *EGY* határozatlan névelő, feltételezhetjük, hogy az *EGY* betűcsoportnak az *SHQ*, vagyis a *G*-nek a *H*, míg a *Q*-nak az *S* betű felel meg… Az eljárást lépésről lépésre folytatva lassan visszafejthetjük a kódoláskor használt betűcseréket, és így megjelenik a *B* oszlopban a titkosított szöveg, Molnár Ferenc Pál utcai fiúk című regénye…

	A	B	C	D	E	F	G
1	**L**	I			**db_{sz}**	**k_{sz}**	**k_m**
2	**V**	H		A	4691	22	123
3	**O**	**A**		B	4368	20	19
4	**N**	R		C	12860	60	10
5	**M**	O		D	0	0	22
6	**W**	M		E	19943	93	125
7	**C**	N		F	11470	53	9
8	**S**	**E**		G	12382	57	38
9	**H**	**G**		H	7739	36	17
10	**Q**	**Y**		I	2238	10	48
11	**S**	**E**		J	73	0	12
12	**A**	D		K	4437	21	45
13	**S**	**E**		L	8814	41	56
14	**H**	**G**		M	16072	75	40
15	**Q**	**Y**		N	8384	39	64
16	**F**	K		O	27369	127	72
17	**M**	O		P	8214	38	8
18	**N**	R		Q	4665	22	0
19	**S**	**E**		R	2471	11	40
20	**U**	P		S	27453	127	59
21	**U**	P		T	2128	10	89
22	**O**	**A**		U	2040	9	18
23	**K**	B		V	3677	17	20
24	**K**	B		W	8102	38	0
25	**O**	**A**		X	11845	55	0
26	**C**	N		Y	3974	18	26
27	**O**	**A**		Z	4	0	39

▶ Titkos szöveg visszafejtése

Feladatok

1. Egy érdekes titkosítási eljárás, amikor az *A*-t *Z*-vel, *B*-t *Y*-nal… cseréljük és fordítva. Írjunk be egy közmondást karakterenként (ékezetek, szóközök és egyéb írásjelek nélkül) az *A* oszlopba, és rejtjelezzük ezzel az eljárással!
2. Töltsük le Mikszáth Kálmán Gavallérok című regényének szövegét a Magyar Elektronikus Könyvtárból! Távolítsuk el a szóközöket, számokat és egyéb írásjeleket, valamint az ékezeteket, majd a betűket cseréljük le nagybetűre és tördeljük új sorba! Rejtjelezzük a kapott szöveget az 1. feladatban leírt eljárással!

Online kommunikáció

Az online kommunikáció szerepe

A tavalyi évben megismerkedtünk az online kommunikáció legfontosabb eszközeivel. Ezek az eszközök átalakították a mindennapi életünket. Segíthetik a közösségi életünket, alkalmasak információszerzésre. Egyre több hivatalos ügyet intézünk az online térben. Tanuláshoz sok esetben online formában használunk tananyagokat, videókat, a kötelező olvasmányok jó része akár elektronikus formában is elolvasható. Időnk egyre nagyobb részét töltjük online környezetben, ezért érdemes átgondolnunk, milyen szabályokat kell betartanunk, milyen veszélyforrásokra érdemes odafigyelnünk.

Digitális lábnyom

Az online kommunikáció során sok olyan tevékenységet végzünk, amelynek eredménye valamilyen formában mentésre kerül. Vannak olyanok, amelyeket a saját gépünkön tárolunk el vagy amelyeket online tárhelyre mi magunk helyezünk el, és olyanok is, amelyeket a meglátogatott webhelyek, közösségi oldalak tárolnak rólunk automatikusan.

A feltöltött képeink, videóink, bejegyzéseink a portálokon megőrződnek, de más adatokat is tárolhatnak rólunk a rendszerek. Ilyenek lehetnek a közösségi oldalakon folytatott tevékenységeink, reagálások, kattintások, megjelölések, a megtekintett videók, a kiadott kereséseink. Akár azt is tárolhatják a rendszerek, hogy mikor, melyik pontján tartózkodunk a világnak, megfigyelhetik a szokásainkat, például, hogy mikor jelentkezünk be, vagy mikor megyünk aludni. Ezek az adatok sokat elárulnak rólunk. Egy-egy meggondolatlanul elhelyezett tartalom, akár sok évvel később felbukkanva kellemetlenségeket okozhat. Az online világban való tevékenységünk során keletkezett, különböző szervereken megtalálható adatok összességét nevezzük *digitális lábnyom*nak. Digitális lábnyomunkat a közösségi oldalak használata és a nem megfelelő biztonsági beállítások nagymértékben növelhetik. Érdemes figyelnünk a digitális lábnyomunk karbantartására. Nézzük át az általunk létrehozott tartalmakat, a rólunk tárolt információkat! Távolítsuk el időről időre a már nem használtakat, és tekintsük át a biztonsági beállításokat! Az általunk elhelyezett tartalmak legyenek olyanok, amelyeket később is vállalhatunk, és amelyek nem okoznak kárt, ha esetleg más birtokába kerülnek. Fontoljuk meg, kinek mihez adunk hozzáférést, korlátozzuk a tartalmak láthatóságát! Óvatosan fogadjunk valakit az ismerőseink közé!

▶ Adatvédelmi beállítások a Facebookon

A felkeresett weboldalak gyakran helyeznek el kis méretű, adatokat tartalmazó fájlokat a számítógépünkön, amelyeket a webszerver és a böngészőprogramok közötti kommunikáció során használnak. Az ilyen adatfájlt **sütinek (cookie)** nevezzük. Szerepük elsősorban az, hogy webhely szolgáltatásait kényelmesebb, testreszabottabb formában vehessük igénybe. A sütik legnagyobb része nem veszélyes, de adatokat tárolnak rólunk, ezért érdemes időnként a feleslegeseket törölni.

Feladatok

1. Nézzünk utána, mit tárolnak rólunk az egyes közösségi oldalak! Keressük meg a biztonsági beállításokat, és tekintsük át!
2. Az alábbi részlet az Instagram biztonsággal kapcsolatos tanácsaiból származik. Keressünk olyan konkrét helyzeteket, amelyekkel szemben az alább javasolt beállítások megvédhetnek bennünket! Beszéljük meg csoportokban!

> - Fontos tisztában lenned azzal, hogy fiókod nyilvános vagy privát fiók-e. Ha a bejegyzéseidet privátra állítod be, ahhoz, hogy valaki megnézhesse bejegyzéseidet, követőidet vagy követési listádat, követési kérést kell küldenie számodra.
> - Ha valaki megfélemlít, kérj segítséget egy olyan családtagodtól vagy tanárodtól, akiben megbízol. Lehetőséged van arra is, hogy eltávolíts egy hozzászólást, amelyet egy általad megosztott fényképhez írtak, illetve hogy jelentsd a megfélemlítést vagy a zaklatást a Súgóközponton keresztül.
> - Ügyelj arra, hogy csak olyan fényképeket és videókat tegyél közzé, amelyek szélesebb közönség (szüleid, tanáraid vagy munkáltatód) előtt sem okoznak kellemetlenséget számodra.
> - Soha ne egyezz bele olyasmibe, illetve olyan tartalom megosztásába, amely kellemetlen számodra.

▶ Biztonsági tippek – Instagram

Viselkedés az online közösségben

A digitális világ használata során be kell tartanunk bizonyos írott és íratlan szabályokat. Ezek a szabályok több forrásból származhatnak. Egy részüket a különböző oldalak használatba vételével, az ott történő regisztrációval elfogadjuk. Ritkán olvassuk végig ezt a szabályrendszert, de ettől függetlenül érvénybe lép, amikor elkezdjük használni az oldal szolgáltatásait.

Az online kommunikáció illemszabályait összefoglaló néven *netikett*nek nevezzük. Az illemszabályok sokfajta helyzetre vonatkozhatnak, és az új kommunikációs formák elterjedésével egyre bővülnek.

Az online kommunikáció során be kell tartanunk a törvényeket. Nem sérthetjük például mások személyiségi jogait, az adatvédelmi szabályokat vagy a szerzői jogokat.

Az online kommunikáció a fogyatékkal élők számára sokszor könnyebb kapcsolattartást ad másokkal. Az online térben nagyobb körből választhatják meg a kommunikáció eszközeit és formáit. Tovább javítják helyzetüket a különböző szoftverekbe beépített kisegítő lehetőségek, például nagyító, virtuális billentyűzet, szövegfelolvasás. Fontos, hogy az egyes helyzetekben rájuk is gondoljunk.

Az online kommunikáció során fennáll annak a veszélye, hogy *online zaklatás (cyberbullying)* valamilyen formáját szenvedjük el. Online zaklatásnak tekintjük, ha valakit az online kommunikációs felületen visszatérően megaláznak, nevetségessé tesznek, fenyegetnek. Az online bántalmazás elől nehezebb elmenekülni, és általában nyilvánosság előtt történik. A bántalmazás elkövetőit bátrabbá teszi, hogy az online térben könnyebb elrejtőzni.

Fontos figyelnünk arra, hogy az online térben is viselkedjünk úgy másokkal szemben, ahogyan fordított helyzetben bennünket sem bántana. Tudnunk kell, hogy mindenkinek más a tűrőképessége. A másik fél érzelmeit az online kommunikáció során kevésbé érzékeljük, ezért sokkal óvatosabban kell viselkednünk. Az online zaklatást nem kell eltűrni! Ha ilyen ér minket, kérjünk segítséget olyan felnőttől, akiben megbízunk!

Kérdések, feladatok

1. Nézzük meg a Tudatosabb internethasználat – Digitális Jólét Program (digitalisjoletprogram.hu) oldalon található Online zaklatás kisfilmet. Beszéljük meg a következő kérdéseket a filmmel kapcsolatban:
   a. Mi jellemző a filmben látott zaklatási helyzetre?
   b. Milyen következményekkel járhat a kialakult helyzet a zaklatottra nézve?
   c. Hogyan lehetne a helyzetet feloldani?
   d. Kitől kérhet segítséget a bántalmazás elszenvedője?

2. Az online zaklatásnak több formája is ismert.
Csoportmunkában keressük meg, milyen online bántalmazási formát jelentenek az alábbi kifejezések! Párosítsuk a fogalmakat a hozzájuk tartozó leírással! Keressünk konkrét példákat a megvalósulásukra!

1 Zaklatás	A Befeketítésre alkalmas pletykák terjesztése valakiről
2 Lejáratás	B Az online közösségből való kirekesztés
3 Lángháború	C Visszatérő támadó, sértő üzenetek küldése
4 Kibeszélés	D Valaki másnak a nevében való posztolás, üzenetküldés
5 Kiközösítés	E Személyes adatok, képek kicsalása, átverés útján történő megszerzése, majd megosztása
6 Identitáslopás	F Személyes információk, titkok megosztása
7 Becsapás	G Dühös, trágár, témába nem illő viták folytatása online felületen
8 Befeketítés	H Valótlan hírek, fotók terjesztése, amelyek lejáratják a másikat

3. Keressünk olyan lehetőségeket, amelyek segíthetik a fogyatékkal élők online kommunikációját! Milyen beállításokat tehetünk az operációs rendszerben, a kommunikációhoz használt programokban (pl. böngésző), amellyel megkönnyíthetjük a munkájukat?

Ajánlott olvasnivaló:
http://dl-sulinet.educatio.hu/download/hirmagazin/cikkek/biztonsagosinternet/www_HU.pdf

Publikálás a világhálón

Korábbi tanulmányainkban statikus honlapokat készítettünk, egy egyszerű kódszerkesztő alkalmazás használatával. A honlap tartalmi elemeit (pl. bekezdések, címsorok stb.) a megfelelő HTML-címkékben (tagekben) helyeztük el, a weboldal megjelenését pedig stíluslapok segítségével állítottuk be. A továbbiakban azzal ismerkedünk meg, hogy hogyan készíthetünk dinamikus weboldalakat, de először ismételjük át a korábban tanultakat!

Ismétlés

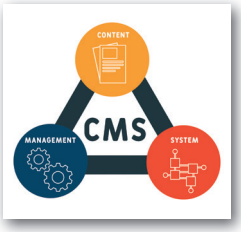
Gyakorlófeladatok segítségével elevenítsük fel, hogy melyek a leggyakrabban használt HTML címkék, és hogy hogyan tudunk stíluslappal alapvető formázásokat beállítani!

Feladatok, kérdések

1. Mit jelölnek az alábbi HTML címkék?
   `<a>`, `<strong>`, `<i>`, `<br>`, `<p>`, `<h1>`, `<h2>`, `<b>`, `<img>`, `<figure>`
2. Milyen stílusbeállításokra alkalmasak az alábbi tulajdonságok?
   `font-family`, `color`, `border`, `text-align`, `background-color`
3. A letölthető állományok között találunk egy *honlap_gyakorlas* nevű mappát. Nyissuk meg a feladatleírást, és a mellékelt állományok felhasználásával készítsük el a statikus honlapot!

Készítsünk dinamikus honlapot tartalomkezelő rendszerrel!

A dinamikus honlapokra az jellemző, hogy egy (webkiszolgálón futó) alkalmazás állítja elő azt a kódot, amelyet a böngészőprogram megjelenít. Ez lehetővé teszi, hogy egy oldalon belül több, különböző forrásból származó tartalom is megjelenhessen. Például egy blog esetén nemcsak a konkrét cikkeket tartalmazhatja az oldal, hanem a közösségi oldalakon beküldött hozzászólásokat, egy hashtag alapján leszűrt képeket, és így tovább. Emellett az oldalak egységesen jelennek meg; minden oldalon azonosak lesznek a fejlécek, a menüstruktúra, illetve a láblécben is ugyanazon információk szerepelnek.



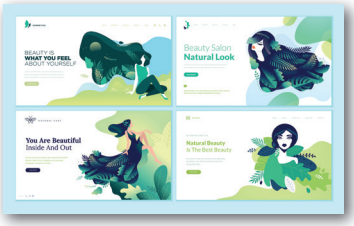
A dinamikus honlapokat gyakran portáloknak nevezik. Ha egy dinamikus honlapot szeretnénk készíteni, akkor érdemes **CMS**-t *(Content Management System)*, vagyis *tartalomkezelő rendszert* használunk. A kifejezetten webes tartalmak előállítására, illetve adminisztrálására szolgáló rendszereket **WCMS**-nek nevezik, amelyben a *W* a *web* (világháló) szóra utal.

A webes tartalomkezelő rendszerek funkciói

A webes tartalomkezelő rendszerek általánosságban a következő funkciókat biztosítják:

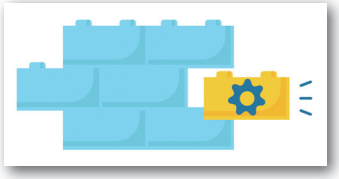
Egyszerű szerkeszthetőség: A szöveges tartalmakat egy egyszerű, vizuális szerkesztőfelület használatával tudjuk megformázni, mintha csak egy szövegszerkesztő alkalmazást használnánk. A beillesztett képeket átméretezhetjük, sőt egyes rendszerekben haladóbb szerkesztési funkciók (felesleges területek levágása, elforgatás, tükrözés stb.) is elérhetők.

Tartalomkezelés: Lehetőség van a különböző tartalmak elmentésére és újbóli szerkesztésére. Ha már elérte a kívánt készültségi szintet az oldalunk, akkor publikálhatjuk azt. Az oldal csak ekkor válik ténylegesen elérhetővé a nagyobb közönség számára. A publikálást visszavonhatjuk, az idejétmúlt cikkeket pedig archiválhatjuk. Természetesen az egyes tartalmakat törölhetjük is, ha már nincs rájuk szükség.



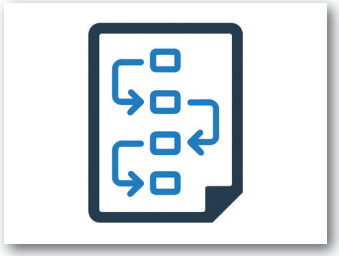
Témák (sablonok), oldalelrendezések használata: az oldal kinézetét a különböző témák vagy sablonok kiválasztásával mi magunk határozhatjuk meg, ezenfelül többféle oldalelrendezést használhatunk a különböző jellegű tartalmak publikálásához. Ez az elrendezés határozza meg, hogy milyen legyen az oldal felépítése, például hogy hány hasábból álljon.

Bővíthető funkcionalitás: A rendszerekhez tartoz(hat)nak olyan beépülő modulok (*plug-in*), amelyeket telepítve újabb és újabb funkciók érhetők el. Például telepíthetünk olyan képgalériát, amely minden eszközön jól használható, reszponzív módon teszi lehetővé a képek, fotók nézegetését, lapozását.



Jogosultságkezelés: A rendszer felhasználóihoz különböző jogosultságokat lehet hozzárendelni. Elképzelhető, hogy egy szerzőnek csak egy tartalmi egység (oldal, bejegyzés) elkészítésére és elmentésére van joga, de azt publikálni nem tudja, mert ahhoz már szerkesztői jogosultságra lenne szüksége. A legtöbb joga az adminisztrátornak van, aki akár testre szabhatja az oldal kinézetét, vagy olyan modulokat telepíthet, amellyel a rendszer tudása bővíthető. Szintén ő az, aki a különböző felhasználókat felveheti a rendszerbe, vagy beállíthatja azt is, hogy szabadon lehessen regisztrálni az oldalra egy adott jogosultsággal, pl. olvasóként.

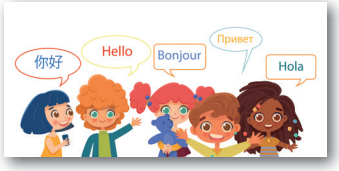
Munkafolyamatok menedzselése: Egy online újság esetén a szerkesztő (illetve a főszerkesztő) dönti el, hogy mikor mely cikkek legyenek publikálva. Az újságíró megírja a cikket, elmenti a rendszerben, amiről a szerkesztő értesítést kap. A szerkesztő visszaküldheti a cikket módosításra, vagy akár ő maga módosíthatja a cikket a publikálás előtt. Vagyis a rendszerben egyszerre több felhasználó tevékenykedik, eltérő szerepekkel. Emiatt fontos, hogy a rendszer jól támogassa a munkafolyamatokat.



Együttműködés: A rendszerek különböző jellegű együttműködést is lehetővé tehetnek a felhasználók között. Elképzelhető, hogy ugyanazt a tartalmat többen állítják elő, egymással összedolgozva.

Verziókezelés: Fontos elvárás, hogy a tartalmak módosításai jól visszakövethetőek legyenek. Emiatt a módosítások után új dokumentumverziók jönnek létre, lehetővé téve, hogy a korábbi változatokat vissza lehessen állítani, például egy véletlenül bekövetkezett törlés vagy felülírás miatt.

Többnyelvűség: A rendszerek biztosíthatják azt, hogy többnyelvű tartalmat állítsunk elő, vagyis hogy a portálnak legyen például magyar, angol és német nyelvű változata is. Ilyen esetben a tartalom feltöltésekor a különböző nyelvi változatokat is fel kell tölteni.



Milyen rendszert válasszunk?

Rengeteg WCMS érhető el a világhálón, ezért először is ki kell választanunk az igényeinknek leginkább megfelelőt. Attól függően, hogy milyen jellegű portált szeretnénk indítani (személyes vagy tematikus weboldal, blog, webáruház), más-más rendszert érdemes választani. Szempont lehet még a rendszer használhatósága, testreszabhatósága, elterjedtsége, korszerűsége, megbízhatósága, ára, a bővíthetőség mértéke, az elérhető támogatás jellege és mértéke (pl. kézikönyvek, fórumok rendelkezésre állása), és így tovább.

Sok olyan rendszer létezik, amelyet akár telepíthetnénk a saját webkiszolgálónkra (webszerverünkre) (pl. Drupal, Wordpress, Joomla stb.), de ehhez és a későbbi adminisztrálásához is megfelelő szakértelemre van szükség. Például gondoskodnunk kell a rendszer megfelelő beállításáról, a rendszeres frissítésről annak érdekében, hogy az oldalunk biztonságosan és megbízhatóan üzemelhessen.

Emellett vannak olyan szolgáltatások is, amelyek online érhetők el. Ebben az esetben regisztrálnunk kell magunkat az oldalon. A szolgáltatásoknak több szintje lehet, amelyeket eltérő áron kínálnak, de elképzelhető, hogy egy bizonyos alapszolgáltatást ingyenesen használhatunk. Az alapszolgáltatások legtöbbször kevesebb funkcionalitást és/vagy tárhelyet biztosítanak, mint a fizetős szolgáltatások.

Az online rendszerek előnye, hogy a környezetet központilag állítják be, és így is frissítik, nekünk tényleg csak a leglényegesebb dolgokra, a tartalom előállítására és publikálására kell koncentrálnunk.

A célunk az, hogy minél egyszerűbben, telepítés nélkül próbálhassuk ki a rendszer lehetőségeit, és hozhassuk létre a honlapunkat, ezért a továbbiakban online elérhető szolgáltatásokkal foglalkozunk.

A honlap létrehozásának és publikálásának lépései

A következőkben bemutatjuk azokat az általános lépéseket, amelyeket a portálunk létrehozása és publikálása során követhetünk.

1. Tartalom létrehozása, összegyűjtése, navigáció megtervezése

Mielőtt a portált létrehoznánk, össze kell gyűjtenünk a publikálandó tartalmat. Ennek során meg kell határoznunk a következőket:

Mi legyen a weboldal elnevezése?

A megadott név jelenik meg az az oldalcímben (`<title>`), illetve sok esetben az oldal fejlécében is. Fontos, hogy a szöveg rövid legyen, és utaljon a weboldal tartalmára. Fontos szerepe van abban is, hogy ha valaki egy webes keresőprogramban rákeres egy kulcsszóra, akkor az oldalunk hányadikként fog megjelenni a találati listában.

Mi legyen a tartalom?

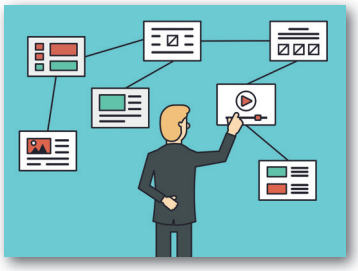
Egy weblap más jellegű médium, mint egy kiadvány. Erre már a tartalom megfogalmazásakor figyelnünk kell. A weblap látogatója nem feltétlenül olvassa el a weboldal teljes tartalmát, hanem pásztázik a szemével. Ez azt jelenti, hogy olyan kulcsszavakat, tartalmi

elemeket keres, amely alapján eldöntheti, hogy számára hasznos-e, érdekes-e az adott oldal. Ezért a szöveget minél lényegre törőbben kell megfogalmaznunk, és ki kell emelnünk a legfontosabb kulcsszavakat, gondolatokat.

Ügyeljünk arra, hogy címsorokkal tagoljuk a tartalmat az oldalon, mégpedig logikus módon. Egyes címsornak csak kettes alcímsorai lehetnek, azoknak hármas szintű alcímsorai, és így tovább. Alakítsuk ki a bekezdéseket, ahol pedig lehet, használjunk listákat a könnyebb áttekinthetőség érdekében.

Egy nyomtatott kiadványban nem helyezhetünk el videót vagy hangállományt, de egy weblapon igen. Használjuk ki a multimédiás elemekben rejlő lehetőségeket, de ne essünk túlzásokba sem!

A web lényege, hogy az oldalak különböző erőforrásokra hivatkozhatnak. Ahol érdemes, állítsunk be olyan hivatkozásokat, amelyek követésével további hasznos információkhoz juthatnak a látogatók. Linkeljük be az oldal elkészítése során felhasznált forrásokat is.



Hogyan tagoljuk a tartalmat, hogyan lehessen navigálni? Ha csak egyetlen oldalból álló, hosszú tartalmat készítenénk, akkor azt kényelmetlenül lehetne használni, illetve az oldal betöltése is lassú lehet. Érdemes a tartalmat kisebb egységekre bontani, és megtervezni a menüstruktúrát. Törekedjünk arra, hogy minél kevesebb menüpontunk legyen. **Almenüket** tervezhetünk, de nem érdemes sokszintű struktúrát használni. Lehetőleg csak egy almenüszint legyen a könnyebb kezelhetőség érdekében, vagyis a főmenünek lehet egy almenüje, de az almenünek lehetőleg ne állítsunk be újabb almenüt, csak akkor, ha egyébként sok (mondjuk hétnél több) menüpont lenne az adott szinten.

Ha olyan oldalt szeretnénk készíteni, amelyre gyakran kerülnek fel hírek, cikkek, bejegyzések, akkor inkább **blog** jellegű oldalban kell gondolkodnunk. Ezek a rendszerek megkülönböztetik az **oldalak**at (amelyek a menüben jelennek meg) a **bejegyzések**től, amelyek jellemzően már a kezdőlapon elérhetők. A bejegyzéseknek jellemzően csak a címük és bevezetőjük látszik a kezdőlapon, a teljes bejegyzések kattintással érhetők el. A bejegyzések különböző **kategóriákba** sorolhatók, illetve akár **címkéket** is hozzájuk lehet rendelni. A bejegyzések az egyes kategóriák, illetve a címkék szerint is szűrhetők. Ha például egy utazással kapcsolatos blogot készítünk, az egyes országok lehetnek kategóriák, így könnyen szűrhetünk egy-egy országra vonatkozó bejegyzésekre. De például elláthatjuk a bejegyzést a *'tengerpart'* címkével is, amelyre szűrve majd a különböző országok tengerparti helyszíneit ismerhetik meg a látogatók. Amikor a híreket (cikkeket) publikáljuk, akkor a megfelelő kategóriákat és címkéket is érdemes beállítanunk.

Természetesen egy portál megtervezésekor akár a megjelenésre (arculatra), az oldalak elvárt elrendezésére (layout) is kitérhetnénk, de ennek megvalósításához már nagyobb szaktudás szükséges, és az ingyenesen elérhető rendszerekben jellemzően nem is tudnánk megvalósítani a terveinket. Viszont az elérhető sémák és elrendezések közül kiválaszthatjuk majd azt, amely számunkra a legmegfelelőbb.

Esettanulmány: Példánkban a Holdról szóló weboldalt fogunk összeállítani, melynek során megismerkedünk a rendszer lehetőségeivel. Hogy a következőkben a tartalom publikálására koncentrálhassunk, a tartalmi egységeket mi már összegyűjtöttük, és meghatároztuk a struktúrát is. A könyvhöz tartozó letölthető állományok között a hold_honlap mappában találjuk a honlap összeállításához szükséges dokumentumokat és képeket.

2. Regisztráció

Azt, hogy a továbbiakban milyen rendszerben kell/lehet dolgoznunk, a tanárunkkal egyeztetve kell meghatároznunk. Elképzelhető, hogy az adott intézmény rendelkezik egy WCMS-sel, amelyhez hozzáférést tud biztosítani a diákok számára, de online elérhető, ingyenes szolgáltatásokat is használhatunk. A szolgáltatás használatához rendelkeznünk kell azonosítóval. Online szolgáltatás esetén jellemzően saját magunkat kell regisztrálnunk. Az alábbi táblázatban két, magyar nyelven is elérhető, népszerű online rendszer adatait találjuk.

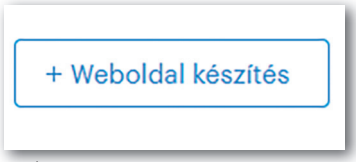
WCMS neve és elérhetősége	Jellemzők	Felület nyelve
Google Sites https://sites.google.com/	Használatához Google-azonosítóval kell rendelkeznünk.	Magyar, angol és más nyelveken is elérhető.
Webnode https://www.webnode.hu/	Tetszőleges e-mail-címmel regisztrálhatunk.	Magyar, angol és más nyelveken is elérhető.

A következőkben bemutatjuk, hogy a regisztráció után milyen teendőink vannak a portálunk elindításáig. Ezen lépések sorrendje csak ajánlott, ezektől esetenként el lehet térni. Előfordulhat, hogy maga a rendszer is megköti az egyes lépések sorrendjét. Lehetséges például, hogy egy adott rendszerben a honlap létrehozása után közvetlenül tudunk csak témát választani, utólag ezt nem változtathatjuk meg. Ezért ennek figyelembevételével kell a folyamatot végigvinnünk.

3. Új webhely létrehozása, alapinformációk megadása

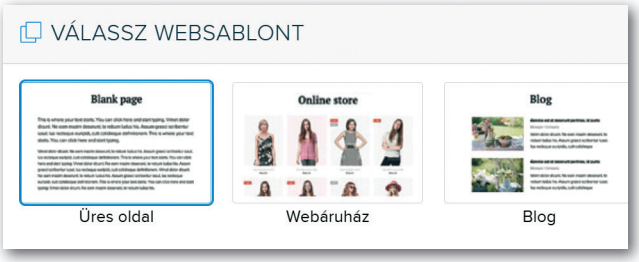
Sikeres regisztráció után egy új webhelyet kell készítenünk a megfelelő ikon/gomb megnyomásával.

► Új webhely készítése ikon (Google Sites)



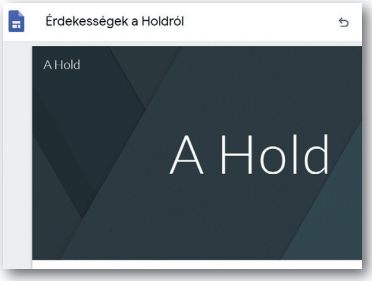
► Új weboldal készítése gomb (Webnode)

Ezek után rendszerint meg kell adnunk a honlapunkra vonatkozó **alapinformációkat**. Egyes rendszerekben ki kell választanunk, hogy milyen jellegű oldalt szeretnénk létrehozni (pl. személyes honlap, webáruház, blog stb.).



► A Webnode az új oldal létrehozásakor a sablonra is rákérdez

Másrészt meg kell adnunk, hogy mi legyen a weboldal neve. A mi esetünkben ez az *„Érdekességek a Holdról”* szöveg lesz.

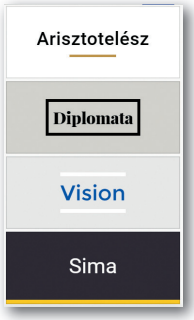


► A webhely nevének (Érdekességek a Holdról) és az oldal címének (A Hold) megadása a Google Sites rendszerben

► A Weboldal nevének megadása (Webnode)

4. Téma (sablon, smink) kiválasztása

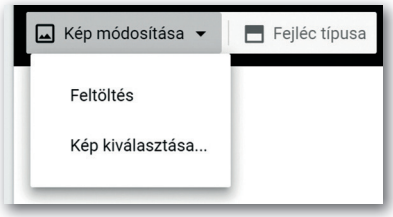
Fontos, hogy olyan témát (sablont) válasszunk ki, amely illik a honlap témájához. Feltétlenül ellenőrizzük le azt is, hogy az adott témát használva helyesen jelennek-e meg a magyar ékezetes betűk, mert sajnos nem biztos, hogy a témához csatolt alapértelmezett betűcsalád támogatja ezt.



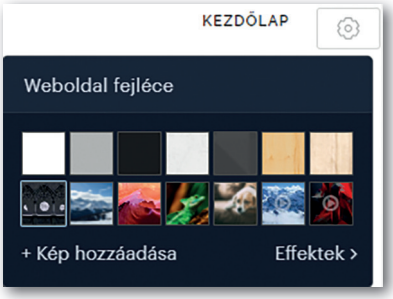
► Google Site-témák

► Webnode-sablonok

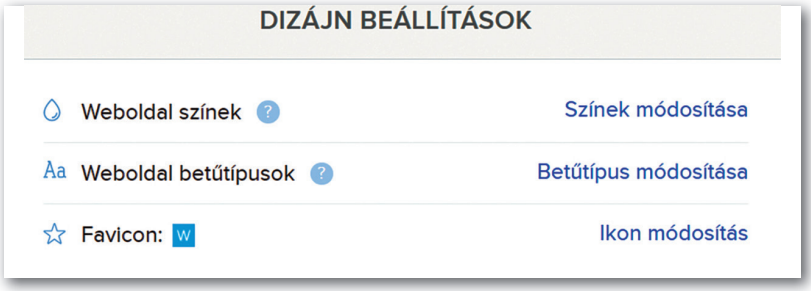
Szintén fontos szempont a téma kiválasztásánál a reszponzivitás biztosítása. Méretezzük át a böngészőprogramot, illetve teszteljük az oldalt mobileszközökön is, hogy meggyőződjünk az oldal megfelelő használhatóságáról. Vizsgáljuk meg azt is, hogy milyen testreszabási lehetőségeink vannak. Elképzelhető, hogy kicserélhetjük a fejléc képét, megváltoztathatjuk a színvilágot, vagy akár a betűtípust.



▶ Fejléc testreszabása (Google Sites)



▶ Fejléc testreszabása (Webnode)

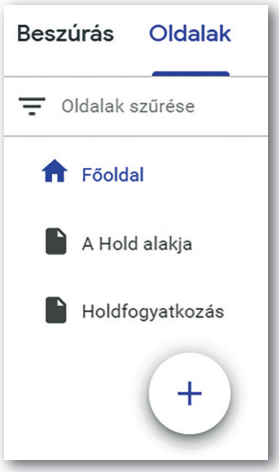


▶ Megjelenés beállításai (Webnode)

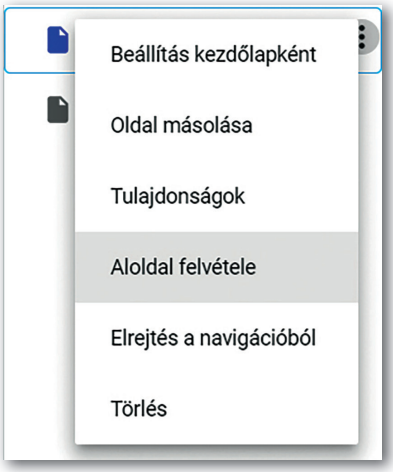
Minél inkább egyedivé varázsoljuk az oldalt, annál kevésbé fog visszaköszönni ugyanaz az arculat a látogatok számára. Arra viszont vigyázzunk, hogy a tartalom mindig jól olvasható legyen, vagyis ügyeljünk a megfelelő kontrasztarányra a háttér és előtér között!

5. Oldalstruktúra meghatározása, oldalak létrehozása

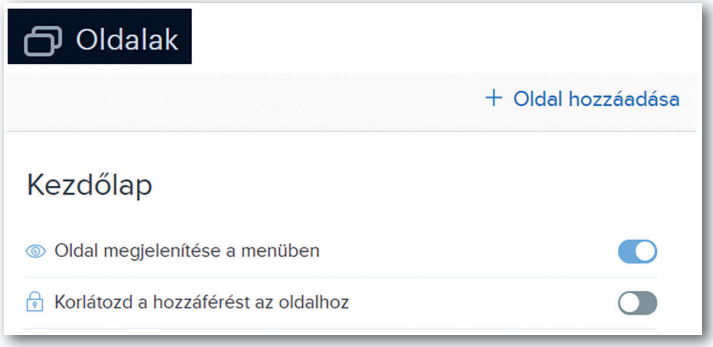
A honlap megtervezésének része az oldalstruktúra meghatározása. Ez kétféle módszerrel történhet a gyakorlatban. Egyrészt kialakítjuk a teljes oldalszerkezetet, majd megkezdhetjük annak tartalommal való feltöltését. A másik módszer, hogy egy-egy oldal tartalmi feltöltése után határozzuk meg, hogy az adott oldal hova kerüljön a menürendszerben. Utóbbi módszernek előnye lehet, hogy publikáláskor nem jelennek meg olyan menüpontok, amelyek még nincsenek feltöltve tartalommal.



► A Google Sites oldalon a + ikonra kattintva tudunk új oldalt létrehozni



► Aloldal létrehozására is van lehetőség az almenüben

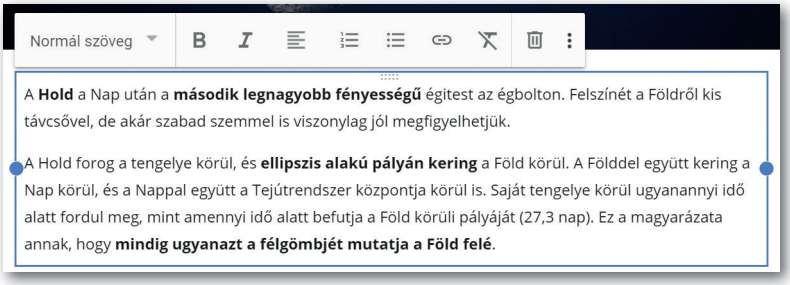


► Oldalak megtekintése és új oldal létrehozása (Webnode)

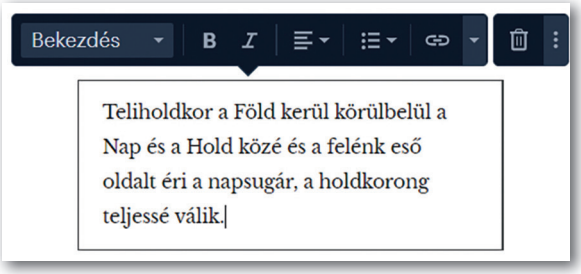
6. Tartalom feltöltése, elrendezés kiválasztása, előnézet

Tartalom feltöltése

Az összegyűjtött tartalmat fel kell töltenünk a tartalomkezelő rendszerbe. A **szövegeket** vágólapon keresztül illeszthetjük be a szerkesztőablakba, majd ezt követően elvégezhetjük az alapvető formázásokat.



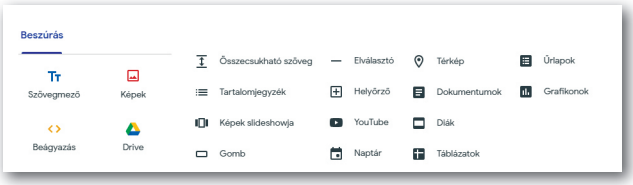
► Szövegszerkesztési lehetőségek a Google Sites alkalmazásban



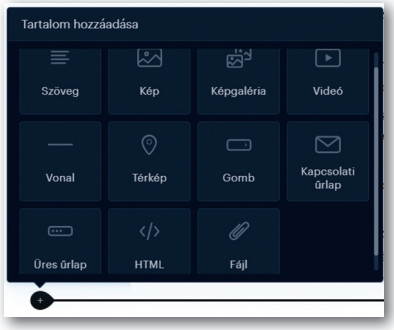
▶ Szövegszerkesztő eszköztár a Webnode alkalmazásban

Képek beillesztése

Szintén fel kell töltenünk a **képeket**, és be kell illesztenünk azokat az oldalakra. A speciális elemek (pl. YouTube-videó, Google-térkép) beágyazására is lehetőséget adnak a korszerű WCMS-ek.



▶ Az oldalra beszúrható elemek a Google Sites oldalán

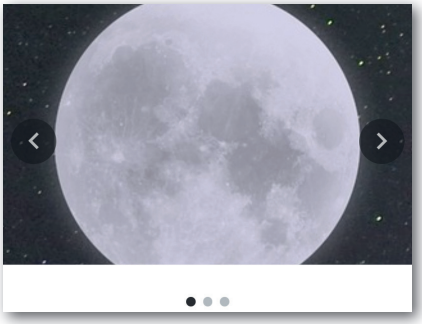


▶ Az oldalra beilleszthető elemek (Webnode)

Képgaléria

Képekből akár egy lapozható galériát is összeállíthatunk.

▶ Galéria szerkesztése a Google Sites alkalmazásban. A + jel megnyomásával bővíthető a galéria.



▶ A böngészőben megjelenő képgaléria, jobbra és balra lapozási lehetőséggel

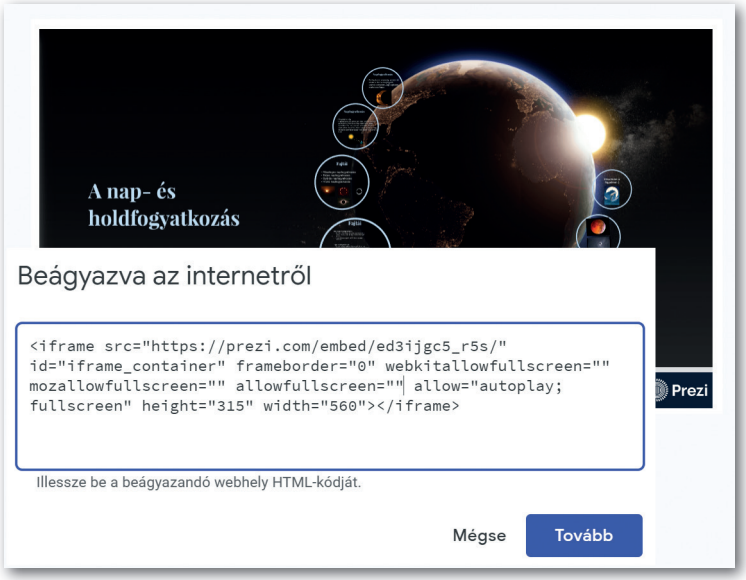
HTML-kód beillesztése

Szintén lehetőség van HTML-kód beillesztésére. Így akár korábban már elkészített HTML-kód-részletet is beilleszthetünk az oldalra, vagy esetleg olyan beágyazókódot is használhatunk, amellyel speciális elemet tudunk az oldalba illeszteni.

Tipp! A https://www.slideshare.net/, illetve https://prezi.com/ oldalon érdekes prezentációkat találhatunk, a legkülönbözőbb témákban, akár magyar nyelven is. Az egyes prezentációkat akár be is ágyazhatjuk az oldalunkra a beágyazókód segítségével. Ez a legtöbb esetben egy `<iframe>` tag segítségével megadott, beágyazott keretet hoz létre az oldalunkon. A prezentációk kiválasztásánál fontos, hogy éljünk a forráskritikával, és valóban csak olyan prezentációkat használjunk fel, amelyek megbízhatóságát ellenőriztük.
Az idézés során használhatjuk a következő sablont.

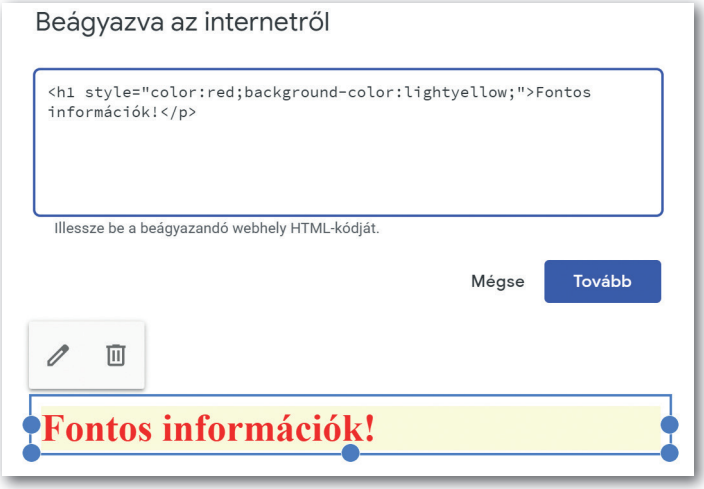
Szerző neve (dátum):	Horváth János (2020):
Hivatkozott anyag címe	Érdekességek a Holdról
Az anyag webcíme	http://tiny.cc/erdekeshold
(Utolsó letöltés: dátum)	(Utolsó letöltés: 2020. 09. 18.)

Ha a szerző neve nem ismert, akkor az idézett anyag címével kezdjük a leírást!



▶ Prezi beágyazókód, mögötte az eredménnyel

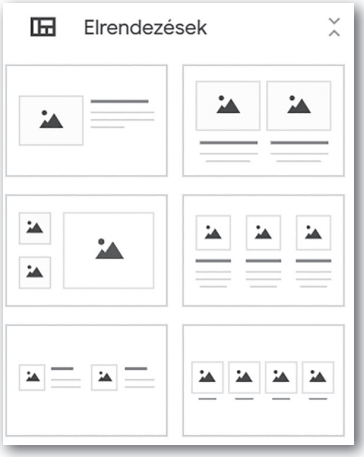
Tipp! Ezzel a módszerrel akár a fejezet elején megoldott gyakorlófeladat kódját is el lehetne helyezni az oldalon. Így olyan stílusok beállítását is elvégezhetjük, amelyeket a WCMS szerkesztője nem támogat. Például megjeleníthetünk adott színnel, háttérszínnel és szegéllyel rendelkező szöveget. Ekkor az adott elemnél használjuk a `style` paramétert, és abban soroljuk fel a tulajdonság-érték párokat!



► Beágyazott kód és annak eredménye az oldalon

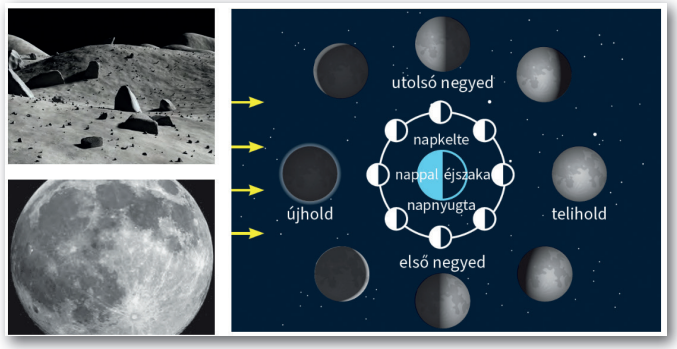
Elrendezés kiválasztása

A tartalomkezelő rendszerek különböző **elrendezéseket** kínálnak fel a tartalom elrendezésére. A fejlett funkcionalitással ellátott felületeken egy oldalon belül akár több szakaszt is létrehozhatunk, amelyekben különböző elrendezéseket használhatunk.

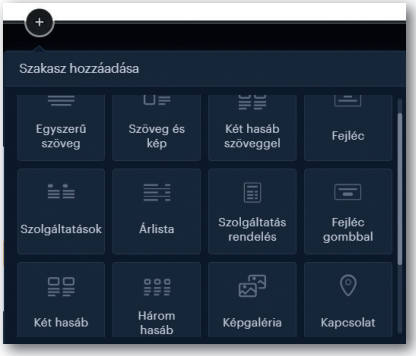


► Elrendezéssablonok a Google Sites portálon

► Három kép címmel és magyarázattal



► Két kisebb és egy nagyobb kép elhelyezése a sablonban



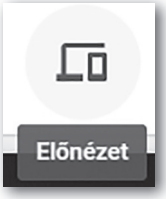
▶ Szakaszon belüli elrendezések a Webnode alkalmazásban

▶ Szöveg és kép, illetve két hasáb elrendezés egymás alatt (Webnode)

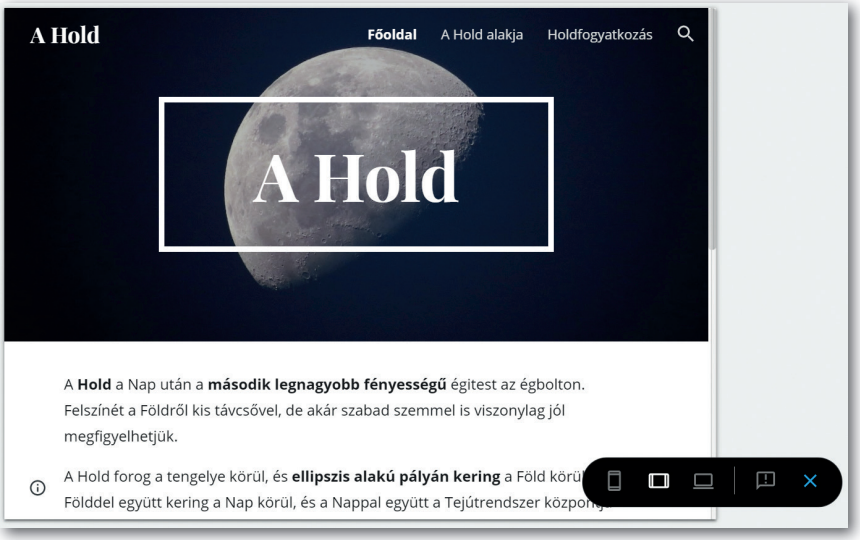
A szakaszokat legtöbbször fogd és vidd módszerrel is áthelyezhetjük az oldalon belül, illetve törölhetjük is őket, ha már nincs rájuk szükségünk.

Előnézet

A tartalomkezelő rendszerek lehetőséget biztosítanak arra, hogy az oldal **előnézetét** megtekinthessük, így publikálás előtt is ellenőrizhetjük, hogy hogyan fog kinézni az adott oldal.



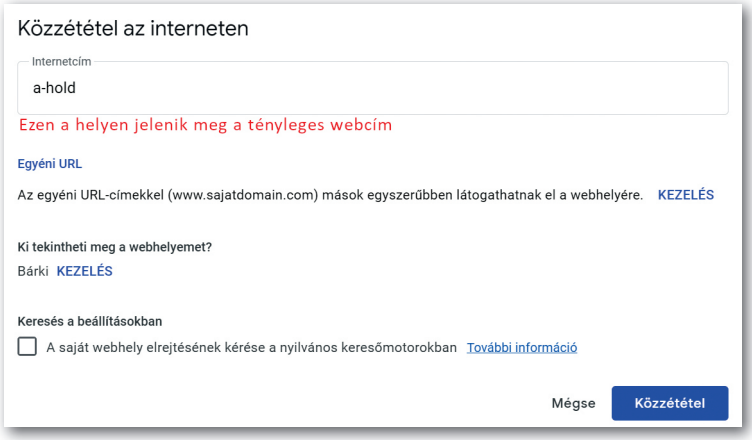
▶ Előnézet ikon (Google Sites)



▶ Az oldal előnézete telefon, táblagép és nagyképernyős megjelenítésre szolgáló ikonokkal (Google Sites)

7. Oldal publikálása

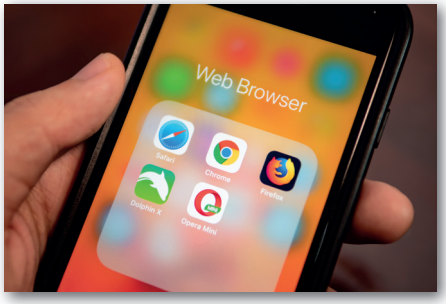
Amíg nem publikáljuk a weboldalt, az csak számunkra érhető el. Ha egy oldal elérte azt a készültségi szintet, amikor már a nagyközönség is láthatja, akkor érdemes közzétennünk (publikálnunk) azt. Az oldal publikálását később visszavonhatjuk.



▶ Közzététel a Google Sites portálon

8. Oldal tesztelése, javítása

Nagyon fontos, hogy az oldalt a publikálás után is teszteljük. Tekintsük meg több böngészőprogrammal az oldalt, vizsgáljuk meg, hogy kisebb és nagyobb felbontásban hogyan jelenik meg. Teszteljük az oldalt okostelefonon is! Ha esetleg problémát tapasztalunk a megjelenítésben, válasszunk más témát és/vagy oldalelrendezést, vagy próbáljuk más módon formázni a tartalmat, például próbálkozzunk egyszerűbb formázásokkal.



Gyakorlás

Itt az idő, hogy a korábban bemutatott folyamatot önállóan is végigvigyük a tartalom összegyűjtésén keresztül egészen a publikálásig.

Feladatok

Válasszunk egyet a következő témák közül (vagy egyeztessünk egy szabadon választottat a tanárunkkal), és készítsük el a honlapot a webes tartalomkezelő rendszerben!

- Bakancslista – a dolgok, amelyeket meg szeretnék tenni az életemben
- Híres magyar találmányok
- Mémek, amelyeket szeretek
- Álmaim autói
- Mi történt a nagyvilágban azon a napon, amikor megszülettem?

- Népszerű összeesküvés-elméletek
- A természet csodái
- Élet a Földön kívül
- Kedvenc
  - hobbim
  - könyvem
  - író/költőm
  - színészem
  - filmem
  - zenéim/előadóim
  - festőm/festményeim
  - történelmi személyiségem
  - tudósom
  - sportágam/sportolóm
  - kirándulóhelyeim
  - koncertem, amelyen részt vettem
  - színházi előadás / kiállítás, amit láttam
  - állatfajaim
  - helyem, ahol már jártam
  - ételeim
  - kütyüim

Elvárások:

- Előre gyűjtsük össze a tartalmat, és tervezzük meg a navigációt! A kezdőlapon kívül legalább két további oldalból álljon a honlap!
- Módosítsuk úgy a sablont, hogy minél inkább utaljon a választott témára!
- A tartalmakat tagoljuk megfelelően, próbáljuk kihasználni minél jobban a szerkesztőfelületen elérhető funkciókat.
- Publikáljuk és teszteljük a munkánkat! A honlap címét osszuk meg a többiekkel, és nézzük meg az ő munkáikat is!
- Adjunk rövid visszajelzést a többiek munkájáról!

Projektmunka

Ha már magabiztosak vagyunk a rendszer használatában, próbáljuk ki azt is, hogy egymással együtt dolgozva, hogyan készíthetünk komplexebb honlapokat! Szerveződjünk három-négy fős csoportokba, és készítsünk egy portált az alábbi témák egyikében vagy a tanárunkkal egyeztetett más témában.

- Más tantárgyakhoz kapcsolódó honlap készítése:
  - Tanultunk valami érdekeset egy másik tantárgyból? Dolgozzuk fel a témakört egy honlap formájában!
- Az iskolai élethez kapcsolódó honlap készítése:
  - Mutassuk meg, milyen lenne az általunk elképzelt iskolai honlap!
  - Nincs még honlapja a diákönkormányzatnak? Nosza, készítsük el!
  - Az osztály szervezett egy remek programot (pl. gólyabált) az iskolában? Készítsünk hozzá honlapot!
  - A Digitális Témahét programjait érdemes lenne külön honlapon bemutatni? Tervezzük meg, és töltsük fel tényleges vagy demó tartalommal!
- Az osztályélethez kapcsolódó honlap fejlesztése:
  - Jól sikerült a legutolsó osztálykirándulás? Állítsunk neki emléket! Mutassuk be a helyszíneket, töltsük fel a fotókat és videókat! Már megvan, hogy hova utazunk idén? Készítsük el a honlapot a helyszínek bemutatásával!
  - Jó lenne egy olyan honlap, amely az egész osztályt bemutatja és folyamatosan frissülne egészen a ballagásig? Itt az ideje lerakni az alapjait!
- Nonprofit szervezet segítése:
  - A közösségi szolgálatot egy szimpatikus, fontos szerepet ellátó helyen teljesítettük, amelynek még nincs honlapja, de nagy szükség lenne rá, hogy többen rátaláljanak? Gyűjtsük össze a szükséges információkat, és lepjük meg őket egy igazán szép és tartalmas honlappal!

A munka során akár eltérő rendszereket is használhatnak a csoportok, majd megoszthatják egymással az ezek használata során szerzett tapasztalataikat.

Projektek bemutatása és értékelése

Az elkészült munkákat mutassuk is be társainknak! A bemutató során térjünk ki a következőkre:

- Milyen szempontok szerint történt a tartalom meghatározása, a struktúra kialakítása?
- Milyen multimédiás tartalmak érhetők el?
- Hogyan sikerült egyedivé tenni az oldalt?
- Hogyan sikerült együtt dolgozni a rendszer által biztosított funkciókkal? Milyen funkció tette volna még hatékonyabbá a munkát?
- Kinek melyik részfeladat volt a kedvence a honlap tervezési és megvalósítási folyamatában?

Az információs társadalom

Gyakran használjuk a kifejezést, **információs társadalom**ban élünk. Mit jelent ez? Milyen jellemzők alapján nevezhetjük így napjaink társadalmát?

Az utóbbi évtizedekben a technikai változások nagy hatással vannak életünk minden területére. A telekommunikációs technológiák gyors fejlődése és elterjedése jelentősen átalakította az életvitelünket, a tanulási formáinkat, a munkavégzésünket. A tudáshoz, az információhoz sokkal könnyebben férünk hozzá, mint korábban bármikor. Ezáltal megismerhetünk olyan helyeket, ahol soha nem jártunk, olyan embereket, akikkel személyesen nem találkoztunk, részt vehetünk olyan eseményeken, vállalkozásokban, amelyek online zajlanak. Egyre több hivatalos és magánjellegű ügyünket intézhetjük hálózaton keresztül, személyes jelenlét nélkül. A társadalom szereplői szempontjából egyre lényegesebbé vált az információ birtoklása, annak minél gyorsabb megszerzése. Fontosabbá válik a tudás, az élethosszig tartó tanulás. Egyre több olyan szakma alakul ki, ahol elengedhetetlen az új kommunikációs technológiák ismerete, rendszeres használata. Aki nem tud lépést tartani a technikai fejlődés miatt megváltozott világgal, hátrányos helyzetbe kerül. Éppen ezért elengedhetetlen, hogy az ezekkel kapcsolatos kihívásokra, változásokra felkészüljünk, eligazodjunk az új eszközök, új lehetőségek között, megfelelően használjuk őket.

A napjainkban is alakuló információs társadalom résztvevőit **digitális állampolgár**oknak nevezhetjük. A digitális állampolgár rendelkezik azokkal a kompetenciákkal, amelyek az információs társadalomban való hatékony részvételhez szükségesek. Digitális állampolgárként végezhetünk olyan értékteremtő tevékenységet, amellyel szűkebb, tágabb környezetünket szolgáljuk. Szükséges, hogy az online térben is betartsuk az etikai szabályokat, figyelembe vegyük az egyéni és közösségi jogokat.

e-szolgáltatások

Az információs társadalom életünk egyre több területére gyakorol hatást. Az átalakuló társadalom lehetőségei és igényei mentén születtek meg a megszokott szolgáltatások új formái, az **e-szolgáltatások**. Egyre szélesebb körben használhatunk ilyen szolgáltatásokat. Nemcsak kapcsolatot tarthatunk a hálózaton keresztül, de intézhetjük bevásárlásainkat, banki ügyeinket, kezelhetjük telefon-, villany- és egyéb szolgáltatásainkat. Egyre több közigazgatással, egészségüggyel és oktatással kapcsolatos szolgáltatást vehetünk igénybe. A rendszerek megfelelő működtetéséhez elengedhetetlen az ügyfelek pontos azonosítása. A magyar állam által fenntartott ügyfélazonosító és -beléptető portál az **Ügyfélkapu**, amelyet a www.magyarorszag.hu címen keresztül érhetünk el. Az ügyek széles skáláját intézhetjük a portálon való bejelentkezést, azonosítást követően, az adóbevallástól a felsőoktatási felvételiig. Egyes államok (például Észtország) már a teljes közigazgatási rendszerüket

az online térbe helyezték át. Az állampolgárok minden hivatalos ügyet intézhetnek ezen a portálon.

A szolgáltatásokkal párhuzamosan az azonosításra szolgáló módszerek is folyamatosan fejlődnek. Sok helyen használjuk még a bejelentkezés legegyszerűbb formáját, amely belépési név és hozzá tartozó jelszó megadását jelenti, de sok egyéb biztonságosabb lehetőség rendelkezésünkre áll már. A közigazgatás által kialakított egyik legfontosabb ilyen azonosító eszköz az **elektronikus személyi igazolvány** vagy **e-személyi**. Az e-személyi nemcsak a hagyományos értelemben vett okmány, hanem a rajta található elektronikus tárolóelem segítségével egyéb személyes adatainkat is tárolja. Ezeket az adatokat az arra jogosult szervezetek megfelelő olvasókulcs segítségével ki tudják olvasni, ezáltal az ügyintézésnél akár személyes jelenlét nélkül is igazolhatjuk magunkat, de sok esetben a személyes ügyintézést is gördülékenyebbé teheti. Az e-személyi igazolvány képes tárolni elektronikus aláírásunkat is.

► Elektronikus tárolóelemet jelző ábra az e-személyi igazolványon

Kérdések, feladatok

1. Keressük fel az Ügyfélkaput (www.magyarorszag.hu)!
   Tekintsük át, milyen ügyeket intézhetnek az állampolgárok erről az oldalról kiindulva!
2. Nézzünk utána, milyen szolgáltatásokat vehetünk igénybe, tehetünk könnyebbé elektronikus személyazonosító igazolványunk segítségével!
3. Alkossunk két-három fős csoportokat. Az alábbi e-szolgáltatásokat osszuk fel a csoportok között:
   a. e-kereskedelem
   b. e-egészségügy, e-recept (előjegyzés, beutalók, leletek kezelése online)
   c. e-közigazgatás (bejelentkezés időpontra, okmányirodai szolgáltatások, adózás, egyéb hivatalos ügyek intézési lehetőségei)
   d. e-bank (banki műveletek online bankban, online fizetés, e-pénztárca)
   e. e-napló szolgáltatásai, online ügyintézés, oktatási szolgáltatások
   f. kultúra, közlekedés e-szolgáltatásai
   g. közmű, távközlési és biztosító cégek e-szolgáltatásai, e-számla-kezelés lehetőségei

   A csoport feladata, hogy tanulmányozza a kapott szolgáltatást!
   Csoportonként töltsük ki a következő táblázatot, majd közösen beszéljük meg, hasonlítsuk össze a szolgáltatásokat!

A csoport témája:	
Milyen előnyei és hátrányai lehetnek a szolgáltatásnak az ügyfél és a szolgáltató szempontjából?	
Előnyök	Hátrányok
Hogyan könnyíti meg az e-szolgáltatás igénybevétele az ügyintézést?	
Van-e olyan lehetőség, ami csak így vehető igénybe, hagyományos formában nem? Mi az?	
Milyen veszélyforrások lehetnek, hogyan lehet ezeket mérsékelni?	
Vannak-e személyes tapasztalataink ezzel az e-szolgáltatással kapcsolatban? Ha igen, melyek azok?	
Milyen fejlődési, fejlesztési lehetőségeket tudunk elképzelni a szolgáltatással kapcsolatban?	

Milyen további fejlődési lehetőségek várhatók az e-szolgáltatások körében? Hogyan képzelhetjük el a jövőt? Nézzünk utána, más országokban milyen lehetőségek vannak!

4. Milyen hatással lehetnek az e-szolgáltatások a globális társadalmi problémák (pl. környezetvédelem, ökológiai lábnyom, esélyegyenlőség, gazdaság teljesítőképessége) megoldására?

Az információs társadalom problémái

Az információs társadalom térhódításával egyre több területen nyílnak meg új, hasznos lehetőségek, de meg kell birkóznunk a gyors és folyamatos változással. Olyan új problémákkal, veszélyekkel kell szembenéznünk, amelyek ezelőtt nem léteztek, vagy nem voltak számottevők.

Digitális személyazonosságunk sok tényezőből tevődik össze. Az online kommunikáció tananyagában már beszéltünk arról, hogy tevékenységeink nyomot hagynak maguk után a hálózaton, és ezek utólag hosszú idő után is követhetők. Ezt nevezzük digitális lábnyomnak. Érdemes ennek tudatában kezelni a feltöltött anyagainkat, megnyilvánulásainkat. Online személyiségünkhöz sok profilt is hozzárendelünk, amelyekkel a különböző szolgáltatások igénybevételekor be kell jelentkeznünk.

Mindannyian tapasztaljuk azt a jelenséget, hogy a különböző weboldalakon elindított információkereséseinknek nyoma marad. Ha valamilyen témában kerestünk, a megjelenő hirdetések ehhez a témához kapcsolódnak, akkor is, ha a keresőoldalról már kiléptünk. Ennek a jelenségnek az okát érdemes egy kicsit megvizsgálnunk. Az egyes weboldalak gyakran figyelembe veszik a megfigyelt böngészési szokásainkat. Ilyenek az elvégzett keresések, az, hogy milyen linkekre, képekre kattintottunk, miket kedveltünk, és mi volt a tartózkodási helyünk. Ezek alapján változik a számunkra megjelenített tartalom. Ez lehet hasznos, ha például vásárolni kívánunk valamit, de lehet káros is, ha például csak azok a vélemények jelennek meg a hírfolyamunkban, amelyek a sajátunkhoz hasonlóak. A fent említett jelenséget **szűrőbuborék**nak vagy **véleménybuborék**nak nevezzük.

Kérdések, feladatok

1. Tekintsük meg az alábbi videót:
   https://www.ted.com/talks/eli_pariser_beware_online_filter_bubbles
   Vitassuk meg a látottakat, beszéljük meg, milyen szempontból veszélyes ez a jelenség!

A világhálón a közösségi oldalakon sok esetben nem az emberek valódi személyiségét ismerhetjük meg. A személyes kontaktus hiánya lehetőséget ad arra, hogy ne a valódi tulajdonságait mutassa valaki. Az anonimitás eredményezheti, hogy olyan megnyilvánulásokat is vállalunk, amelyet személyazonosságunk ismeretében nem tennénk meg.

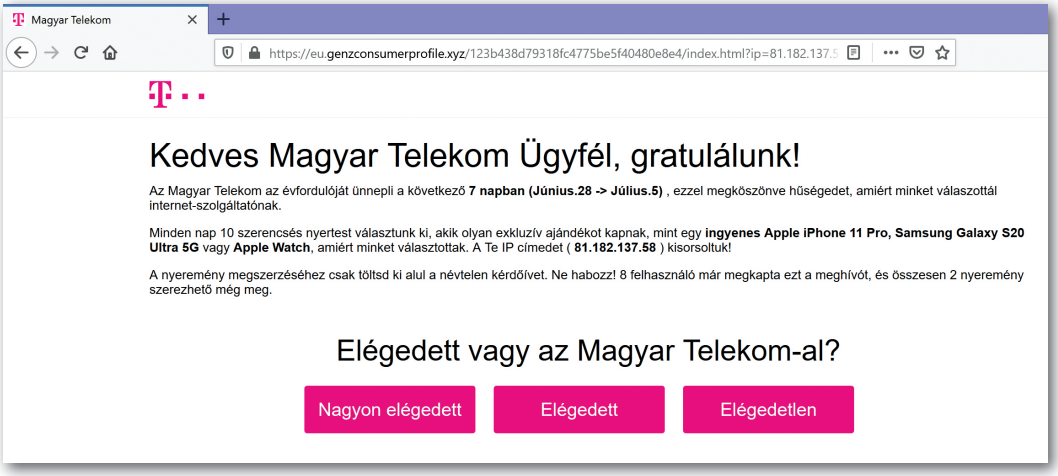
Az interneten gyakran találkozhatunk olyan információkkal, amelyeknek hitelessége megkérdőjelezhető. A valóságalapot nélkülöző, sokszor szenzációként megjelenő híreket **álhír**nek (fake news) nevezzük. Léteznek olyan, magukat hírportálként feltüntető oldalak, amelyek azért hoznak létre ilyen bejegyzéseket, hogy minél többen rákattintsanak, és ezzel növeljék a saját látogatottságukat, bevételüket. Lehet az oldal célja valamilyen politikai vagy más társadalmi jellegű megtévesztés. Az ilyen megtévesztő hírek másik típusa e-mai-

len vagy más üzenetközvetítő platform segítségével terjed, esetleg kártékony programrészt is tartalmazhat. Ezeket nevezzük **hoax**nak.

A megtévesztő tartalmak egy másik csoportjának az a célja, hogy valamilyen fontos adatunkat megszerezze, vagyis az **adathalászat**. Az ilyen átverő tartalmú üzenetek, weboldalak egy-egy szolgáltató weboldalának, hivatalos üzeneteinek megtévesztő utánzatai, amelyekben fontos adataink, például bankszámlaszám, PIN-kód, jelszó megadását kérik tőlünk. Az átverést nem minden esetben lehet könnyen észrevenni, legyünk figyelmesek, óvatosak!

Kérdések, feladatok

2. Hogyan vehetjük észre egy hírről, hogy álhír? Milyen jelek lehetnek, amelyek alapján erre gyanakodhatunk? Keressük fel az alhirvadasz.hu oldalt, ahol segítséget kaphatunk a kérdések megválaszolásához!
3. Az alábbi ábrán egy hamis tartalmú oldal látható. Milyen jelek segíthetnek abban, hogy felismerjük az átverési szándékot?



Az informatikai eszközökkel végzett kommunikáció során nem csak a kapott információk hitelességével kell foglalkoznunk. Saját adataink között is vannak olyanok, amelyek védelmére oda kell figyelnünk. Azokat az adatokat, amelyek alapján azonosítani lehet azt a személyt, akire az adatok vonatkoznak, rá nézve következtetéseket lehet levonni, **személyes adat**nak nevezzük. A természetes személyek személyes adatainak védelme, a magánélet sérthetetlensége alkotmányos alapjog. Ezt Magyarország Alaptörvénye és az Európai Unió Alapjogi Chartája is szabályozza. Az információs társadalom a személyes adatok védelmét is jelentősen átalakította. Adataink, köztük a személyes adatok, az online környezetben sokkal sebezhetőb-

bek, könnyebben hozzáférhetők lettek. Ez az oka, hogy szükségessé vált az adatvédelmi szabályok átalakítása. Az Európai Unió 2016-ban alkotta meg új rendeletét a természetes személyek személyes adatainak védelméről (angolul: **G**eneral **D**ata **P**rotection **R**egulation, rövidítve: **GDPR**). A rendeletet 2018 májusától kell alkalmazni. Az online kommunikáció során is tekintettel kell lenni ennek rendelkezéseire. Hatásait a mindennapi tevékenységeink közben gyakran tapasztaljuk.

A törvényi védelem mellett magunknak is tennünk kell azért, hogy személyes adatainkat biztonságban tartsuk. Az online kommunikáció fejezetben már beszéltünk a közösségi portálok adatvédelmi beállításairól. Online tevékenységünk során egyre több és több helyre kell bejelentkeznünk. A különböző regisztrációkat gyakran összekötjük e-mail-fiókunkkal vagy közösségi portálon lévő profilunkkal, ezért egy-egy regisztráció feltörése több adatunkhoz is elvezethet. A bejelentkezéshez használt jelszavainkra fontos odafigyelnünk. Nem célszerű több helyen ugyanazt a jelszót használni, kevéssé biztonságos jelszavakat választani, mert ez is sérülékennyé teheti az adatainkat. Sok bejelentkezéssel kapcsolatosan kérhetjük a **többfaktoros azonosítást**. Ilyenkor a jelszavunk megadása mellett más típusú azonosítót is kér tőlünk a rendszer (például ujjlenyomat, kulcskártya). Azzal a lehetőséggel is gyakran élhetünk, hogy ezt csak abban az esetben kérje tőlünk a portál, ha a körülmények megkívánják. Ilyen lehetőséggel rendelkezik például a Google és a Facebook is. Ha ismeretlen eszközről vagy szokatlan helyről próbálunk bejelentkezni, egy megbízhatónak beállított eszközünkre megerősítő kódot küld, és csak ennek megadásával tudjuk befejezni a bejelentkezést.

Összefoglalva: az információs társadalom világa sok szempontból kitárta előttünk az ajtót, sok értelemben eddig nem tapasztalt szabadságot, új lehetőségeket hozott az életünkbe. Ennek megvannak az előnyei, de szembe kell néznünk az általa okozott nehézségekkel, veszélyekkel, igyekeznünk kell kiküszöbölni azokat. Érdemes körültekintően, megfontoltan viselkednünk az online térben.

Kérdések, feladatok

4. Milyen ismérvei vannak a jó jelszónak? Milyen stratégiát választhatunk annak érdekében, hogy a jelszavaink biztonságosak legyenek, de mégse okozzon gondot a számontartásuk? Beszéljük meg közösen, ha szükséges, nézzünk utána!
5. Hogyan, milyen helyzetekben találkozhatunk a GDPR-rendeletnek megfelelő adatkezelésssel? Milyen jeleit vehetjük észre?

Algoritmizálás és programozási nyelv használata

Eddig jutottunk eddig

Könyvünk előző kötetében belekóstoltunk a programozásba, és elég sok mindent megtanultunk – először ezeket ismételjük át.

Mindenhol programok

Tudjuk már, hogy a háztartási gépektől kezdve az autókon és a repülőgépeken keresztül a robotokig mindenütt van számítógép, és ahol számítógépek, ott programok is vannak. A digitáliskultúra-órákon a bennünket jobban érdeklő, hagyományos értelemben vett számítógépek (laptopok, asztali gépek, szerverek) és mobileszközök a bekapcsolásukkor egy fő programot indítanak el, az operációs rendszert.

A többi program elindítása, futásuk közben az eszköz erőforrásaihoz (perifériák, háttértárak, memória, processzor) való hozzáférés szabályozása és a programok megállítása az operációs rendszer feladata. A programok egy része automatikusan indul, más részüket a felhasználó indítja el.

Kérdések

Milyen operációs rendszer fut a számítógépeden és milyen a mobileszközeiden?
Milyen háttértár van a számítógépedben, milyen a mobileszközeidben?

A programok elindításukig csak a háttértáron találhatók meg. Elindításkor a memóriába tölti őket a számítógép, és a processzor megkezdi a végrehajtásukat. A programot tartalmazó fájl természetesen megmarad a háttértáron ilyenkor is. A programok elindítása történhet az ikonjukra való kattintással, az ikon ujjunkkal történő megérintésével, de minden program elindítható a számítógép parancssorából is.

Nyissuk meg a számítógépünk parancssorát, és indítsunk el innen egy szövegszerkesztőt, egy képszerkesztőt, egy böngészőprogramot!

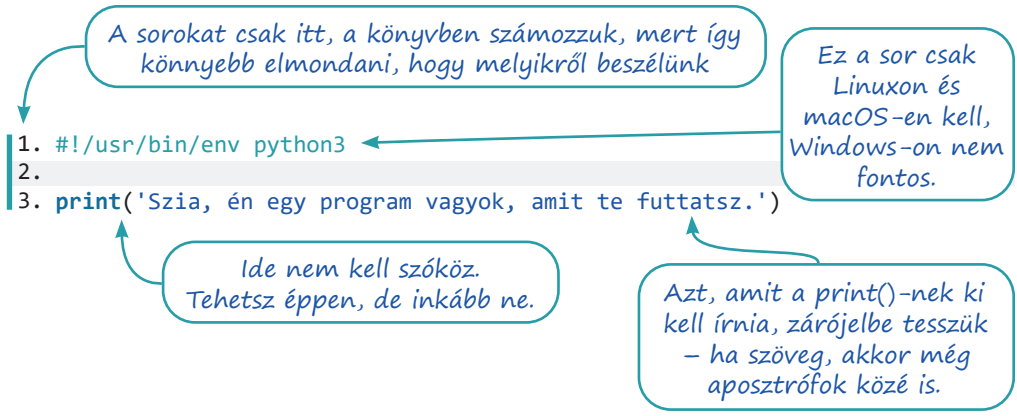
Forráskód, programozási nyelvek és fejlesztői környezet

Programjainkat az esetek túlnyomó többségében forráskódként fogalmazzuk meg. A forráskód az angol nyelvből vett szavakon kívül rendszerint szép számban tartalmaz még mindenféle egyéb jeleket és számokat, és hellyel-közzel mindenki el tudja ezeket olvasni – a programozáshoz értők nyilván lényegesen eredményesebben.

A forráskódot sima szöveges fájlokban tároljuk, és a fájl kiterjesztése általában a programozási nyelvre utal. Egy köszönést a képernyőre író egyszerű program neve Ruby nyelv használata esetén lehet `szia.rb`, C++ nyelven dolgozva a fájlnév alighanem a `szia.cpp` formát ölti, és ha – mint ahogy az előző kötetben és ebben is – Pythont használunk, akkor a fájl neve esélyesen `szia.py`.

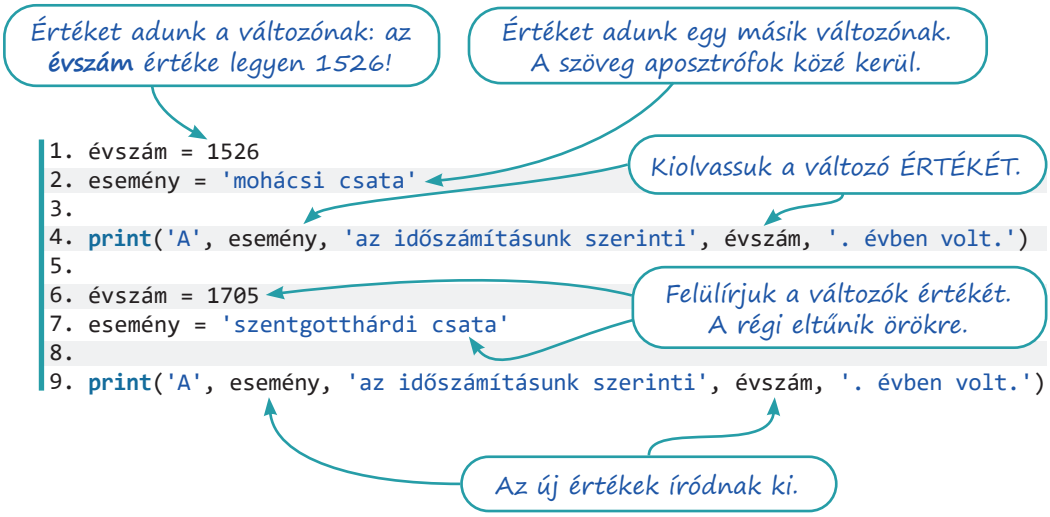
A programjainkat a legegyszerűbb szerkesztőprogramban is írhatjuk, de általában valamilyen fejlesztői környezetet, azaz IDE-t használunk. A legegyszerűbb IDE-ket „csak” az különbözteti meg az egyszerű szerkesztőktől, hogy színezéssel segítik a programozót a programban való jobb eligazodásban. Vannak ennél lényegesen nagyobb tudású és ennek megfelelően bonyolultan használható IDE-k is.

Nyissunk meg egy IDE-t, és írjuk meg benne azt a programot, amelyik elárulja, hogy épp egy programot futtatunk:



Változók

A programjainknak gyakran kell adatokat tárolniuk. Az adatokat a gép memóriájába teszik el, és hogy a memórián belül pontosan hova, azt a legtöbbször nem tudjuk. Az eltett adatokat úgy tudjuk ismét elővenni, ha megadjuk azt a nevet, amit az eltett adathoz hozzárendelünk. Ezt a hozzárendelt nevet változónak hívjuk, és az eltárolást programozóul úgy mondjuk, hogy értéket adunk a változónak. Az értékadás egy művelet, ugyanúgy, mint az összeadás vagy az osztás, és van műveleti jele is, ami sok programozási nyelvben – a Pythonban is – az egyenlőségjel.



A fenti kód kimenete csúnyácska, hiszen amikor a `print()` a vesszővel elválasztott kiírandó elemeket összefűzi, szóközt is tesz, és így a pont nem kerül pontosan az évszám mellé. A megoldást is ismerjük már, hasonlítsuk össze az alábbi két sort:

```
1. print('A', esemény, 'az időszámításunk szerinti', évszám, '. évben volt.')
2. print('A ', esemény, ' az időszámításunk szerinti ', évszám, '. évben volt.', sep='')
```

Ez itt két aposztróf, köztük pedig nincs semmi.

A második sor végén, a `sep=''` részben, a két aposztróf közé kerül az a karaktersorozat, amit a `print()` a vesszővel összefűzött részek közé ír. Alapértelmezetten ez egy szóköz, de mi most semmit sem írunk közéjük, azaz nincs elválasztó karakter, és a pont végre az évszám mellé kerül. Viszont így sehol máshol sem lesz szóköz a sorban, nekünk kell beírni mindenütt.

Változók típusai, típusátalakítás, adatbekérés

A felhasználótól adatot az `input()` utasítással kérhetünk be. A bekért eredményt változóban tárolhatjuk. Az `input()` nem olyan ügyes, mint a `print()`:, nem tud összefűzni több kiírandó részt. Szerencsére az összeadást a Python karaktersorozatokra is tudja értelmezni: az `'el' + 'iramlik'` művelet eredménye `'eliramlik'`. Az összeadással azonban baj lehet, ha egy számot és egy szöveget adunk össze. Mennyi a `2 + 'asztal'`?

Az `str()` (string, azaz karaktersorozat) utasítással a zárójelben lévő számot szöveggé alakíthatjuk, és `'2' + 'asztal'` már a Python számára is `'2asztal'`.

Ez is értékadás: amit az input() a felhasználótól kap, azt betesszük a szorzó nevű változóba.

```
1. szorzandó = 5
2. szorzó = input('Mennyivel szorozzam meg az ' + str(szorzandó) + '-öt? ')
3. print(szorzandó, '-ször ', szorzó, ' annyi mint ', sep='', end='')
4. szorzó = int(szorzó)
5. print(szorzandó * szorzó)
```

Itt azt mondjuk meg a print()-nek, hogy még ne kezdjen új sort.

Amit az input() a felhasználótól kap, azt mindig szövegként adja át, még a számokat is. Ha matekozni akarunk vele, akkor például egész számmá (integer, azaz int) kell alakítanunk.

Eddig összesen négyféle típusú elemi adatot tároltunk változóban:

- karaktersorozatot, más néven szöveget;
- egész számot;
- logikai értéket (az ilyen változók értéke `True` [igaz] vagy `False` [hamis] lehet);
- és ritkábban tizedestörtet, más néven lebegőpontos számot (persze tizedesponttal a vessző helyett).

Elágazások

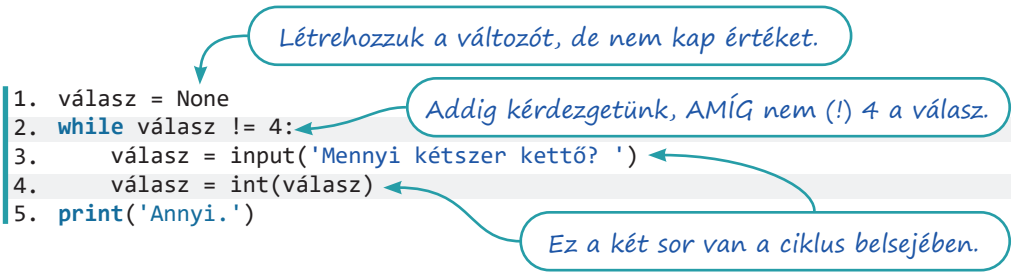
Nagyon hamar felmerül az igény, hogy a programunk eltérő feltételek esetén másként viselkedjen. Például, ha elmúlt este nyolc, váltson sötét témára a telefon, ha helyesen adta meg a jelszót a felhasználó, akkor engedjük belépni. Az ilyen problémák megoldására való az `if` és az `else` utasítás.

Mit csinál az alábbi program?


```
1. ellenség = input('Ki volt a Piroska nevű szuperhős főellensége? ')
2. if ellenség == 'farkas' or ellenség == 'Farkas':
3.     print('Okos vagy.')
4.     print('Nem is kicsit.')
5. else:
6.     print('Hááát....')
7.     print('Nem.')
8. print('Legközelebb a hét törpét kérdezem - visszafelé!')
```

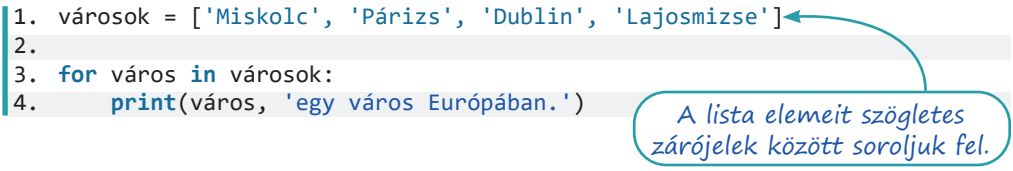
Ciklusok és listák

Ha egy feladatrész ismétlődik, akkor ciklus használatával oldjuk meg. Van feltételes és bejárós ciklusunk. A feltételes ciklus magja addig ismétlődik, amíg fennáll a ciklus elején megfogalmazott feltétel. Az amíg (angolul `while`) a feltételes ciklus elejét jelző utasítás.




```
1. válasz = None
2. while válasz != 4:
3.     válasz = input('Mennyi kétszer kettő? ')
4.     válasz = int(válasz)
5. print('Annyi.')
```

Az összetartozó adatokat listában tároljuk, a listákat pedig bejárós ciklussal tudjuk bejárni. A bejárós ciklus ciklusváltozója (a lenti példában a város) mindig a lista aktuális elemét tartalmazza. A bejárós ciklus másik neve: `for`-ciklus.




```
1. városok = ['Miskolc', 'Párizs', 'Dublin', 'Lajosmizse']
2.
3. for város in városok:
4.     print(város, 'egy város Európában.')
```

Elemi adattípusok és elágazások

Feladatok

1. Írjuk képernyőre programmal egy általunk választott vers két versszakát! A vers előtt adjuk meg a szerzőt és a címet, majd sorkihagyást követően az első, újabb sorkihagyást követően a második versszakot írjuk ki! A programunk legfeljebb három `print()` utasítást használhat.
2. Mondatszerű leírással (más szóval: pszeudokódban) megadunk egy programot. A program bemenete egy állatfaj és az állat legnagyobb sebessége km/h-ban kifejezve.

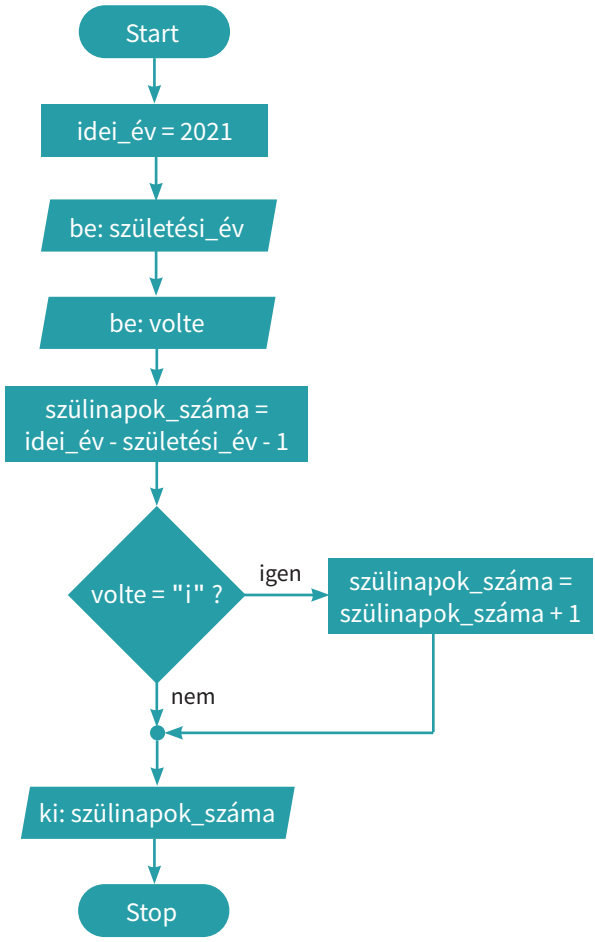
```
program
    be: állatfaj
    be: sebesség
    elágazás
        ha sebesség legfeljebb 50:
            hol := „városban”
        különbenha sebesség legfeljebb 90:
            hol := „országúton”
        különben:
            hol := „autópályán”
    elágazás vége
    ki: „Az”, állatfaj, „a legnagyobb sebességével”, hol,
        „haladhat.”
program vége
```

a. Mit csinál a program?
b. Írjuk át a mondatszerű leírást folyamatábrává!
c. Kódoljuk a programot!

```
állatfaj = input('Add meg egy állatfaj nevét! ')
sebesség = input('Add meg az állatfaj sebességét! ')
sebesség = int(sebesség)
if sebesség <= 50:
    hol = 'városban'
elif sebesség <= 90:
    hol = 'országúton'
else:
    hol = 'autópályán'
print('Az', állatfaj, 'legnagyobb sebességével', hol, 'haladhat.')
```

d. Teszteljük a program működését az interneten fellelhető, a témába vágó adatok használatával!
e. Nagyon hamar találunk olyan állatot, amely esetében hibás ítéletet hoz a programunk. Mit kell tudnia az ilyen állatnak? Hogyan javítható a programunk?

3. Kérdezzük meg a felhasználótól, hogy melyik évben született, és hogy volt-e már idén születésnapja! A két adat ismeretében írjuk ki, hogy hányadik születésnapját ünnepelte! Először készítsük el a megoldás folyamatábráját vagy mondatszerű leírását. Ha elkészültünk, írjuk meg a programkódot is.



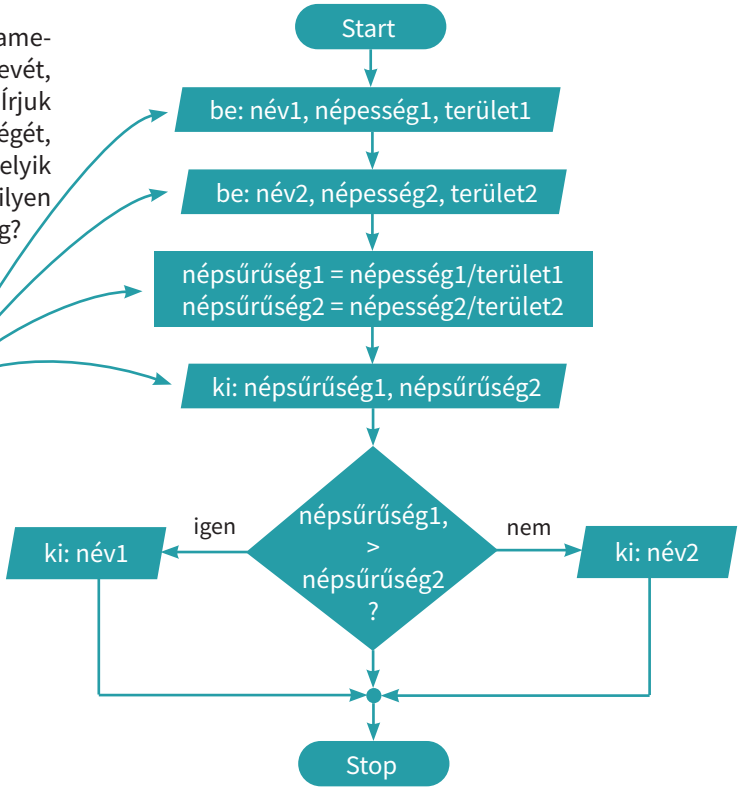

```
program
    idei_év = 2021
    be: születési_év
    be: volte
    szülinapok_száma = idei_év – születési_év - 1
    ha volte = 'i':
        szülinapok_száma = szülinapok_száma + 1
    ki: szülinapok_száma
program vége
```

```
1. idei_év = 2021
2. születési_év = input('Melyik évben születtél? ')
3. születési_év = int(születési_év)
4. volte = input('Volt már idén születésnapod? (i/n) ')
5.
6. szülinapok_száma = idei_év - születési_év - 1
7. if volte == 'i':
8.     szülinapok_száma = szülinapok_száma + 1
9. print('Utoljára a ', szülinapok_száma, '. születésnapodat ünnepelted.', sep='')
```

4. Kérdezzük meg a felhasználótól a nevét és azt, hogy a nap hányadik órájában járunk! Köszönjünk neki a nevét megemlítve a napszaknak megfelelően, reggel nyolcig „Jó reggelt”, este hatig „Jó napot”, aztán „Jó estét” kívánva!
   a. Készítsük el a program mondatszerű leírását vagy folyamatábráját!
   b. A leírás vagy az ábra alapján kódoljuk a programot!
   c. Kihívást jelentő feladat: Ne a felhasználótól kérdezzük meg az órát, hanem olvassuk ki a számítógép órájából! Az internet segíteni fog az óra kiolvasásának kódolásában.
5. Írjunk olyan programot, amelyik bekéri két autómárka nevét és az autók maximális sebességét! Írjuk ki, hogy melyik autó a gyorsabb.

```
program
    be: egyik_neve
    be: egyik_sebessége
    be: másik_neve
    be: másik_sebessége
    elágazás – ha egyik_sebessége > másik_sebessége:
        ki: egyik_neve
    különbenha másik_sebessége > egyik_sebessége:
        ki: másik_neve
    különben:
        ki: 'Egyformán gyorsak.'
    elágazás vége
program vége
```

6. Írjunk olyan programot, amelyik bekéri két ország nevét, népességét és területét! Írjuk ki a két ország népsűrűségét, és azt, hogy ez az adat melyik országban a nagyobb! Milyen típusú adat a népsűrűség?



Nem baj, ha a folyamatábrán az utasításokat alkalmasan csoportosítjuk. Az a lényeg, hogy értsük az ábrát!

7. Kérjünk be két egész számot a felhasználótól, és írjuk ki, hogy a kisebb osztója-e a nagyobbnak (azaz egész szám-e a hányados)!
   a. Készítsük el a feladatot megoldó algoritmus mondatszerű leírását!
   b. Kódoljuk az algoritmust: készítsük el a programot!
   c. A példamegoldás csak pozitív egészekkel boldogul. Módosítsuk úgy akár a példát, akár saját működő programunkat, hogy negatív számokat is megadhassunk!
   d. Módosítsuk úgy a programunkat, hogy csak a valódi osztókat minősítse osztónak!
   e. Írjuk meg a megoldást úgy, hogy az alapműveletek közül csak az összeadást és kivonást használhatjuk.

A bekért számot azonnal egésszé alakítjuk. Ha tetszik a megoldás, használd bátran!

```
1. egyik = int(input('Mi legyen az egyik szám? '))
2. másik = int(input('Mi legyen a másik szám? '))
3.
4. # mindig a nagyobb szám legyen az osztandó
5. if egyik >= másik:
6.     osztandó = egyik
7.     osztó = másik
8. else:
9.     osztandó = másik
10.    osztó = egyik
11.
12. hányados = osztandó / osztó
13. if hányados == int(hányados):
14.     print(osztó, ' osztója ', osztandó, '-nek.', sep='')
15. else:
16.     print(osztó, ' nem osztója ', osztandó, '-nek.', sep='')
```

A kettőskereszttel kezdődő sort a Python figyelmen kívül hagyja. Így írunk megjegyzéseket a kódba.

Ha a szám egésszé alakítva ugyanaz marad, akkor már eleve egész volt.

Mint a legtöbb programozási nyelvben, a Pythonban is van olyan osztás, amelyik a maradékot adja meg. Ez az úgynevezett moduloosztás, vagy egyszerűbben csak mod, és Pythonban a jele a %. Így írjuk le a műveletet: 9 % 4 = 1, azaz kilencet néggyel osztva egy a maradék. A fenti program 12–13. sora helyett tehát használható az

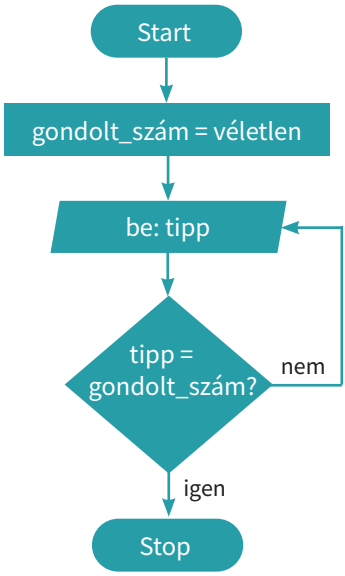
```
if osztandó % osztó == 0:
```

sor is.

Ciklusok és listák

Feladatok

1. Gondoljon a programunk egy számra egy és tíz között! A felhasználó feladata a szám kitalálása. Addig próbálkozhat, amíg el nem találja. A program folyamatábrával és mondatszerű leírással:




```
program
    gondolt_szám := véletlen(10)
    tipp := speciális érték
    ciklus amíg tipp <> gondolt_szám:
        be: tipp
    ciklus vége
program vége
```

Ide valami olyan értéket írunk, ami biztosan NEM a jó megoldás. Lehet pl. –1, de Pythonban a None a bevett szokás.

a. Hogyan állítunk elő véletlen számot? Keressük meg az interneten (angolul több találatunk lesz, keressünk a *random integer Python* kifejezésre)!
b. Kódoljuk az algoritmust!

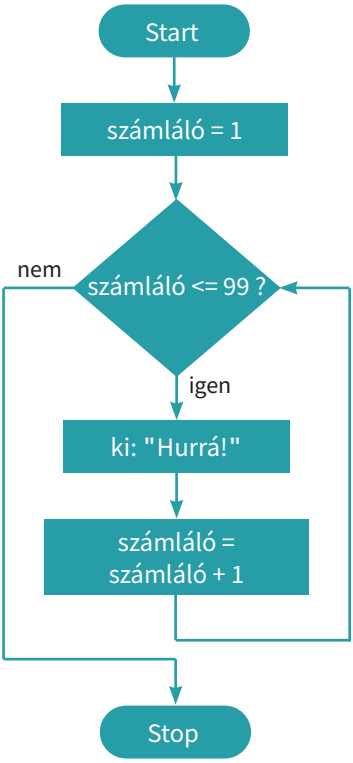
```
1. import random
2.
3. gondolt_szám = random.randint(1,10)
4. tipp = None
5. while tipp != gondolt_szám:
6.     tipp = input('Tippelj! ')
7.     tipp = int(tipp)
```

c. Javítsunk annyit a programon, hogy találat esetén dicsérje meg a felhasználót!
d. Módosítsuk úgy a programot, hogy írja ki a felhasználónak, hogy a hibás tipp kisebb vagy nagyobb a gondolt számnál!

2. Vegyünk fel a programunkba egy listát: `['barack', 'körte', 'dinnye', 'narancs']`! Írjuk ki bejárós ciklussal mindegyikről, hogy egy gyümölcs!

3. Írjuk ki kilencvenkilencszer, hogy „Hurrá!”

   a. A kiírást először feltételes, azaz `while`-ciklussal végezzük el. Szükségünk lesz egy számláló nevű változóra, aminek a ciklusba való belépés előtt az 1 értéket adjuk, majd minden kiírást követően növeljük az értékét a cikluson belül. A ciklusba való belépés feltétele az, hogy a számláló értéke ne legyen nagyobb 99-nél.




```
számláló = 1
ciklus amíg számláló <= 99:
    ki: „Hurrá!”
    számláló := számláló + 1
ciklus vége
```

```
1. számláló = 1
2. while számláló <= 99:
3.     print('Hurrá!')
4.     számláló = számláló + 1
```

   b. Megoldjuk a feladatot bejárós, azaz `for`-ciklussal is. A `for`-nak be kell járnia valamit, ami 99 elemű – ilyen bejárható objektumot készíthetünk a `range(99)` utasítással. Az utasítás használatát keressük meg interneten!

```
ciklus 99-szer:
    ki: „Hurrá!”
ciklus vége
```

A mondatszerű leírásban élhetünk alkalmas egyszerűsítéssel – elvégre csak nekünk kell érteni!

```
1. for _ in range(99):
2.     print('Hurrá!')
```

Pythonban az alávonást szokás ciklusváltozóként használni, ha a változót a ciklusmagban nem használjuk. Ez csak szokás, nem szabály!

   c. Írjuk át mindkét programunkat úgy, hogy ne legyen sortörés a szavak után, csak miután kiírtuk mind a kilencvenkilencet!

4. Írjuk ki az egy és száz közötti hárommal osztható számokat! Elegáns megoldás volna a `for szám in range(3,101,3)` utasítás használata, de most tekintsünk el tőle – így többet tanulunk az algoritmusokról.

   a. Írjuk meg a programot mondatszerű leírással!

   b. Kódoljuk a programot! Segít, ha tudjuk, hogy létezik a maradékképző osztás, aminek a jele: „%”. A művelet eredménye az osztás maradéka, azaz
   9%3 = 0 és 9%2 = 1

c. Számoljuk meg, és írjuk ki, hogy hány hárommal osztható számot találunk!
d. Módosítsuk úgy a programot, hogy egy és száz helyett a felhasználó által megadott számokat használjuk!
e. Írjuk át úgy a programot, hogy a felhasználó mondhasson számot a három helyett!
f. Írjuk meg a programot a másik ciklus felhasználásával!

A megoldás bejárós ciklussal:

```
1. eleje = input('Honnan induljunk? ')
2. eleje = int(eleje)
3. vége = input('Meddig számoljunk el? ')
4. vége = int(vége)
5. osztó = input('Hánnyal osztható számokat keresünk? ')
6. osztó = int(osztó)
7. ennyi_ilyen_van = 0
8.
9. for szám in range(eleje,vége+1):
10.     if szám % osztó == 0:
11.         print('A', szám, 'osztható ezzel:', osztó)
12.         ennyi_ilyen_van = ennyi_ilyen_van + 1
13.
14. print(ennyi_ilyen_van, 'ilyen számot találtam.')
```

A feltételes ciklust használó megoldásban a 9-től 12-ig számozott sorokat cseréljük ki az alábbiakra:

```
9. szám = eleje
10. while szám <= vége:
11.     if szám % osztó == 0:
12.         print('A', szám, 'osztható ezzel:', osztó)
13.         ennyi_ilyen_van = ennyi_ilyen_van + 1
14.     szám = szám + 1
```

5. Állítsunk elő egy ezer és tízezer közötti egész számokat tartalmazó, húszelemű listát! A lista elemei azoknak a járműveknek a tömegét adják meg, amiket ma egy komphajó átvitt a folyón. Nehéznek számítanak a 9300 kilónál nehezebb járművek. Írjunk programot, ami válaszol a következő kérdésekre:
   a. Volt-e olyan jármű ma a hajón, ami nehéznek számít? Írjuk ki, ha volt ilyen!
   b. Hány ilyen jármű volt?
   c. Hány kiló járművet vitt át a komp ma összesen?
   d. Mennyi a ma átvitt, nehéznek számító járművek össztömege?
   e. Ha a „nehéz” holnaptól nem 9300, hanem 9000 kilogramm, hány helyen kell átírni a programot? Mit kell tennünk, ha azt szeretnénk, hogy az ilyen változások egyszerűen, egyetlen helyen való átírást jelentsenek?

```
import random

tömegek = []
for _ in range(20):
    tömegek.append(random.randint(1000,10000))
# kész a listánk
print(tömegek) #csak hogy könnyű legyen ellenőrizni

volt_nehéz = False
nehezek_száma = 0
össztömeg = 0
nehezek_össztömege = 0
for tömeg in tömegek:
    össztömeg = össztömeg + tömeg
    if tömeg > 9300:
        volt_nehéz = True
        nehezek_száma += 1
        nehezek_össztömege += tömeg

if volt_nehéz:
    print('Volt 9300 kilónál nehezebb jármű.')

print(nehezek_száma, '9300 kilónál nehezebb jármű volt.')
print('Ma', össztömeg, 'kilónyi járművet vitt át a komp.')
print('Ebből nehéznek összesen', nehezek_össztömege, 'kilónyi számít.')
```

Ha nem használjuk a ciklusváltozót, adhatunk neki semmitmondó nevet.

A változók értékét többféleképp is növelhetjük.

Jó úgy is, hogy
If volt_nehéz == True:
de így könnyebben olvasható.

Szövegek, eljárások, függvények

Mit tudunk eddig a szövegekről?

Szövegeket (karakterláncokat, stringeket) már használtunk programjainkban. Azt is tudjuk, hogy két szöveg összeadható egy összeadásjellel, és az összeadás valójában összefűzést, egymás mellé írást jelent. A csak számokat tartalmazó karakterláncokból tudunk egész vagy valós számot csinálni (az `int`() vagy a `float`() utasítással), és a számokat is át tudjuk alakítani szöveggé (az `str`() utasítással).

Karakterláncok mint listák

A szövegeket a Pythonban szinte minden esetben kezelhetjük úgy, mintha karakterekből álló listák lennének. Lássunk rá egy példaprogramot!

```
1. szöveg = 'szöveg'
2. lista = ['l', 'i', 's', 't', 'a']
3.
4. #kiírás
5. print(szöveg)
6. print(lista)
7. print(''.join(lista))
8.
9. #első és utolsó elem
10. print(szöveg[0], szöveg[-1])
11. print(lista[0], lista[-1])
12.
13. #milyen hosszú a string, hány elemű a lista
14. print(len(szöveg))
15. print(len(lista))
16.
17. #bejárásuk, egy-egy szóközzel, hogy látványosabb legyen
18. for karakter in szöveg:
19.     print(karakter, ' ', end='')
20. print('')
21.
22. for elem in lista:
23.     print(elem, ' ', end='')
24. print('')
```

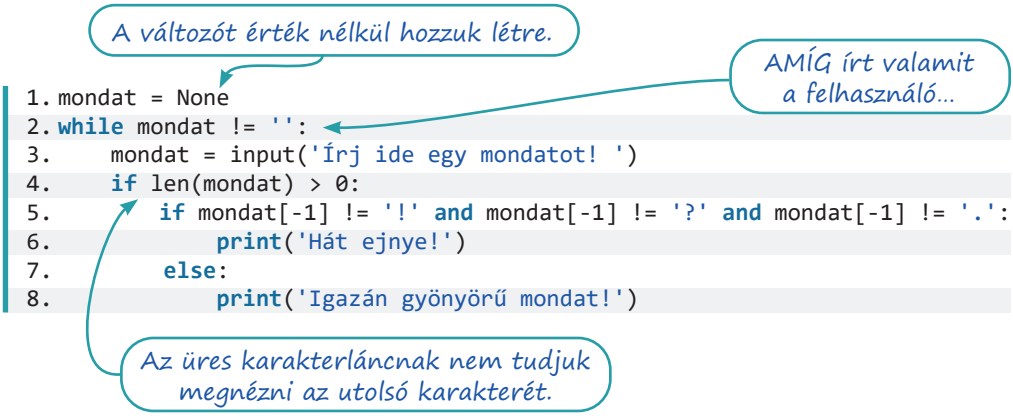
Nincs semmi a két aposztróf között.

Van persze még egy csomó közös tulajdonságuk, és vannak eltérések is – idővel megismerkedünk velük.

Milyen esetben szidja össze az alábbi program a felhasználót?

```
1. mondat = input('Írj ide egy mondatot! ')
2. if mondat[-1] != '!' and mondat[-1] != '?' and mondat[-1] != '.':
3.     print('Hát ejnye!')
4. else:
5.     print('Igazán gyönyörű mondat!')
```

Írjuk át úgy a programunkat, hogy addig kérjen új meg új mondatokat, amíg a felhasználó meg nem elégeli a mókát, és üres bemenetet nem ad (azaz nem ír be semmit, csak lenyomja az Entert)!

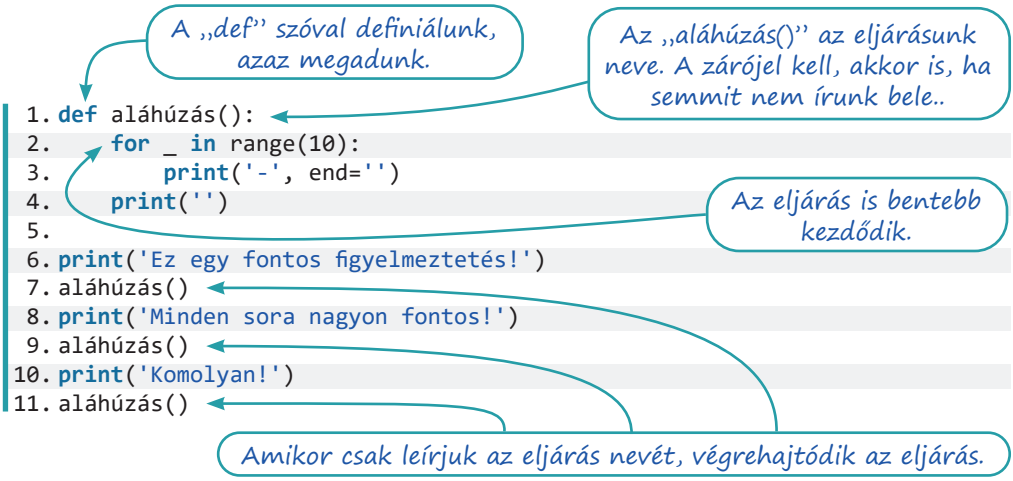



```
1. mondat = None
2. while mondat != '':
3.     mondat = input('Írj ide egy mondatot! ')
4.     if len(mondat) > 0:
5.         if mondat[-1] != '!' and mondat[-1] != '?' and mondat[-1] != '.':
6.             print('Hát ejnye!')
7.         else:
8.             print('Igazán gyönyörű mondat!')
```

A ciklusmag már így is szép nagy, pedig bővíthetnénk is a programunk tudását. Úgyhogy megtanuljuk a programjainkat részekre szedni.

Eljárást írunk

Eljárást két esetben ír a fejlesztő (azaz mi), és mindjárt megbeszéljük, hogy mi ez a két eset. Először azonban meg kell értenünk, hogy mi az eljárás. Vegyünk elő megint egy példaprogramot! A program mindössze annyit tesz, hogy kiír háromsornyi szöveget, aláhúzva. Persze a parancssorban nem tudunk aláhúzott betűket írni, így az „aláhúzás” valójában egy új sor, benne kötőjelekkel. Az eljárás az első négy sorban van.




```
1. def aláhúzás():
2.     for _ in range(10):
3.         print('-', end='')
4.     print('')
5.
6. print('Ez egy fontos figyelmeztetés!')
7. aláhúzás()
8. print('Minden sora nagyon fontos!')
9. aláhúzás()
10. print('Komolyan!')
11. aláhúzás()
```

Amikor a Python azt olvassa a programunkban, hogy `def`, akkor tudja, hogy most figyel, de nem csinál semmit. A `def` után következő részt majd máskor kell végrehajtania. Amikor viszont csak az eljárás nevét olvassa, `def` nélkül (először a 7. sorban látunk ilyet), akkor

1. megkeresi az ilyen nevű eljárást (igen, több eljárás is lehet egy programban),
2. végrehajtja az eljárást,
3. visszajön oda, ahonnan elment végrehajtani az eljárást, és folytatja a programot.

Ha kipróbáltuk a programunkat, akkor megbeszéljük, hogy eljárást két esetben használunk:

- amikor a programunkban több helyen is ugyanaz van, akkor az ismétlődő részt kiszervezzük egy eljárásba,
- de az is lehet, hogy egy helyen van csak egy rész, mégis kitesszük eljárásba, mert úgy könnyebben olvasható a program.

Mire kellenek a zárójelek?

Tegyük fel, hogy többféle aláhúzást is szeretnénk, mondjuk csillagokból állót, meg hullámvonalakból készültet. Akkor most írjunk három eljárást `sima_aláhúzás()`, `csillagos_aláhúzás()` és `hullámos_aláhúzás()` néven? Csak van valami jobb módszer! És tényleg van: a paraméteres eljárás.

Írjuk át az eljárásunkat így:

Ez az eljárás paramétere (vagy argumentuma).

```
1. def aláhúzás(jel):
2.     for _ in range(10):
3.         print(jel, end='')
4.     print('')
```

Itt használjuk fel a paraméter értékét.

Még javítsunk három helyen a programunkban: a hetedik sor legyen ilyen: `aláhúzás('-')`, a kilencedik ilyen: `aláhúzás('*')`, az utolsó meg ilyen: `aláhúzás('~')`. Ha lefuttatjuk a programunkat, látjuk, hogy minden aláhúzás más.

A programunk ilyenkor a következőt csinálja:

1. Amikor az eljárás neve után talál valamit a zárójelben, azt *átadja* az eljárásnak.
2. Kikeresi, hogy az eljárás neve után milyen szó áll (náluk ez volt a `jel`). Ez lesz a paraméter neve, kicsit olyan, mint egy változónév – a továbbiakban ki lehet olvasni az értékét.
3. Az eljárás belsejében ezzel a névvel hivatkozunk az átadott értékre (nálunk a harmadik sorban ezért írja a program azt a karaktert, amit átadtunk az eljárásnak).

Függvényt írunk

A függvény ugyanúgy a program egy elkülönült, magában is értelmezhető része, mint az eljárás. Ugyanabban a két esetben használjuk, mint az eljárást. Ugyanúgy paramétert lehet neki átadni, mint az eljárásnak. Pythonban ugyanazzal a `def` szóval kezdődik a megadása. Akkor mi a különbség? A kimenete pontosan megegyezik. Alaposan hasonlítsuk össze a két kódot!

```
1. def pluszkettő(szám):
2.     print(szám+2)
3.
4. print('5+2= ', end='')
5. pluszkettő(5)
6. print('4+2= ', end='')
7. pluszkettő(4)
```

```
1. def pluszkettő(szám):
2.     return (szám+2)
3.
4. print('5+2= ', pluszkettő(5))
5. print('4+2= ', pluszkettő(4))
```

A bal oldali kódban eljárással valósítjuk meg a feladatot. Az eljárás paraméterében mondjuk meg, hogy melyik számhoz kell kettőt adni. Amikor az eljárást *hívjuk* (azaz leírjuk a nevét, 5. és 7. sor), akkor lefut. Elvégzi az összeadást, és az eredményt kiírja, mert erre utasítja a `print()`. A futás végeztével a kód végrehajtása visszakerül az eljárás hívásának helyére.

A jobb oldalon függvénnyel valósítjuk meg a feladatot. A függvény a hívását követően lefut. Elvégzi az összeadást, visszatér a hívás helyére, az eredményt pedig *visszaadja* a főprogramnak. Olyan, mint ha a függvény futása utáni pillanatban a függvény által visszaadott érték kerülne a függvény nevének helyére, először a 4., majd az 5. sorban. A kiírást a főprogramunk végzi.

Eljárás vagy függvény?

Eljárás és függvény között tehát az a különbség, hogy a függvénynek van visszatérési értéke, az eljárásnak nincs. A visszatérési értéket a függvényben a `return` szót követően adjuk meg.

Az eljárást és a függvényt a legtöbb programozási nyelv nem különbözteti meg élesen, és csak a függvény szót használja. Az ilyen nyelvek – mint a Python is – és az ilyen nyelveken fejlesztők az eljárásokra esetleg csak visszatérési érték nélküli függvényekként tekintenek.

Vissza a kezdetekhez!

Ha újra elővesszük a mondatos programunkat, akkor látjuk, hogy a mondat megítélése egyszerűen kiszervezhető eljárásba. Említettük, hogy eljárásokat két esetben írunk: ha valamit többször kell végrehajtani, vagy pedig olvashatóbbá válik a főprogram. A mondatos programban az utóbbi okból használunk eljárást.

```
 1. def megítél(mondat):
 2.     if len(mondat) > 0:
 3.         if mondat[-1] != '!' and mondat[-1] != '?' and mondat[-1] != '.':
 4.             print('Hát ejnye!')
 5.         else:
 6.             print('Igazán gyönyörű mondat!')
 7.
 8. mondat = None
 9. while mondat != '':
10.     mondat = input('Írj ide egy mondatot! ')
11.     megítél(mondat)
```

Eljárások a gyakorlatban

Egy lényeges megfontolás

Mind az eljárások, mind a függvények megtervezésekor a legtöbbször érdemes úgy dolgoznunk, hogy

- az eljárásunk, illetve függvényünk ne nyúljon a főprogram változóihoz (nemhogy ne próbálja őket átírni, de olvasni se akarja),
- hanem csak a számára híváskor átadott értékekkel dolgozhat.

Vannak programozási nyelvek, ahol erre erősen figyelnünk kell, a Pythonban azonban sokat kell ügyeskedni, ha az eljárás vagy függvény belsejéből babrálni akarunk a főprogram változóival. Szerencsére kifejezetten nem akarunk, úgyhogy ez jó így nekünk.

Feladatok

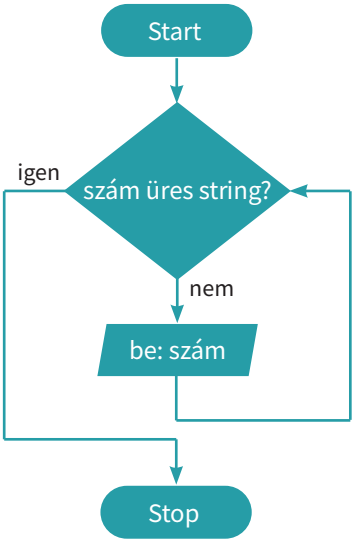
1. Írjuk meg azt az eljárást, amely a számára átadott, literben kifejezett térfogatot átváltja akóba, és az eredményt képernyőre írja! Hogy egy akó hány liter, azt keressük ki az internetről! Sokféle akó van, használjuk a nekünk tetszőt! Hívjuk az eljárást úgy a főprogramból, hogy kiderüljön, hogy 999 liter hány akó!

```
eljárás akóba_vált(liter)
    ki: liter/58,6
eljárás vége

akóba_vált(999)
```

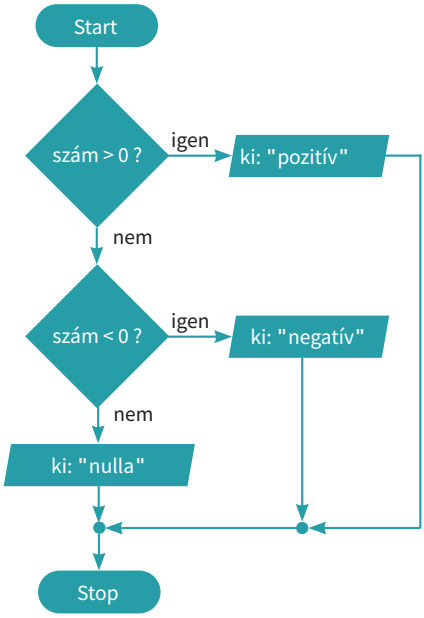
```
1. def akóba_vált(liter):
2.     print(liter/58.6)
3.
4. akóba_vált(999)
```

2. Írjunk olyan programot, amely számokat kér a felhasználótól, amíg üres bemenetet nem kap, majd eljárással kiírja, hogy az épp beírt szám pozitív, negatív vagy nulla!
   a. Először írjuk meg a főprogramot, azaz azt a részt, amely bekéri a számokat (de még ne csináljunk semmit a számmal)!




```
1. szám = None
2. while szám !='':
3.     szám = input('Írj ide egy számot! ')
```

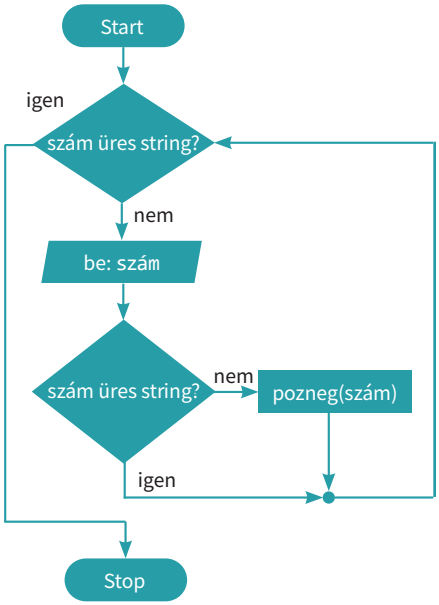
b. A folyamatábra vagy a mondatszerű leírás alapján kódoljuk az eljárást! Az eljárások, függvények nevének megválasztásakor is érdemes figyelnünk arra, hogy a kódot olvasó embert a név segítse a kód megértésében.




```
eljárás pozneg(szám)
    elágazás
    ha szám > 0:
        ki: szám, 'pozitív.'
    különbenha szám < 0:
        ki: szám, 'negatív.'
    különben:
        ki: 'A szám nulla.'
    elágazás vége
eljárás vége
```

```
1. def pozneg(szám):
2.     if szám > 0:
3.         print(szám, 'pozitív.')
4.     elif szám < 0:
5.         print(szám, 'negatív.')
6.     else:
7.         print('A szám nulla.')
```

c. Helyezzük el az eljárást hívó részt a főprogramban! Az eljárás nincs felkészülve rá, hogy üres bemenetet kapjon – ha ilyet kap, a program hibaüzenettel kilép. A főprogramnak figyelnie kell arra, hogy csak akkor hívja az eljárást, ha a bemenet nem üres.




```
1. szám = None
2. while szám != '':
3.     szám = input('Írj ide egy számot! ')
4.     if szám != '':
5.         szám = float(szám)
6.     pozneg(szám)
```

3. Megírandó programunk – igencsak sztereotip döntéseket hozó – bébiszittert szimulál. A program megkérdezi a gyerekek nevét, majd a lányoknak babát, a fiúknak autót ad játszani. A főprogram dolga, hogy neveket kérdezgessen, amíg üres bemenetet nem kap. Egy eljárás dönti el a gyerekek nemét az alapján, hogy a nevük benne van-e a `lánynevek` vagy a `fiúnevek` listában. Ugyanez az eljárás írja ki, hogy az adott gyerek mit kapott játszani. A gyerek nevét az eljárás paramétereként adja át a főprogram.
Azt, hogy valami benne van-e egy listában, nagyon egyszerű eldöntenünk. Mutatunk rá egy példát:

```
1. lista = ['egyik elem', 'másik elem']
2. vizsgált = 'harmadik elem'
3. if vizsgált in lista:
4.     print('Benne van.')
5. else:
6.     print('Nincs benne.')
```

Az „in” ugyanúgy műveleti jel (más szóval: operátor), mint a „+” vagy a „*”.

a. Írjuk meg a programot!
b. Egészítsük ki a programot úgy, hogy ha egyik listában sem szerepel a név, akkor kérdezze meg, hogy a gyerek milyen nemű!
c. A `random.choice()` függvény bevetésével oldjuk meg, hogy mind a fiúk, mind a lányok számára több játékból is válogasson a program!

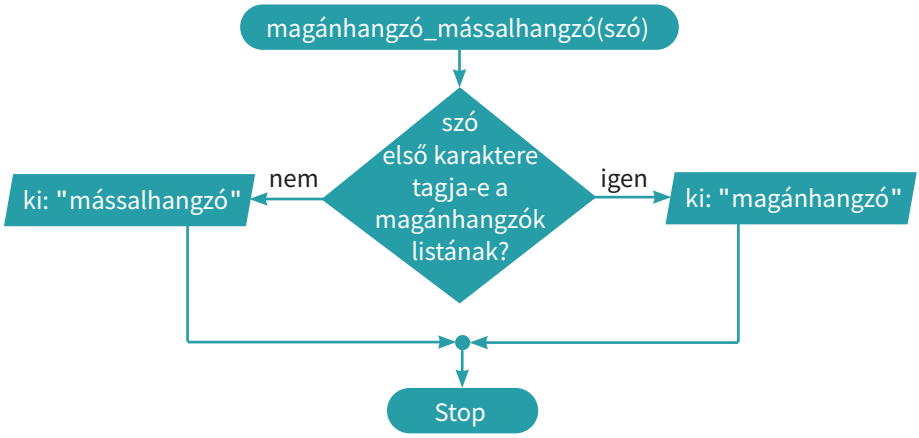
4. Írjunk olyan programot, amelyik listába gyűjti a felhasználó által megadott városokat, majd a listát átadja egy eljárásnak! Az eljárás megszámolja és kiírja, hogy hány európai főváros van az átadott listában.
a. A program megírását az eljárás megírásával kezdjük, mégpedig azért, mert így sokkal egyszerűbb tesztelni az eljárás működését. Az eljárás mondatszerű leírása:

```
eljárás hány_főváros(városok):
    fővárosok = [’Párizs’, ’Riga’, ’Budapest’, … ]
    fővárosok_száma = 0
    ciklus városok minden város-ára:
        ha város eleme fővárosok-nak:
            fővárosok_száma = fővárosok_száma + 1
    ciklus vége
    ki: fővárosok_száma, ’főváros van a listában.’
eljárás vége
```

Kódoljuk az eljárást!
A teszteléskor a főprogram egyetlen sor, például:
`hány_főváros(['Gárdony', 'Zágráb', 'Nagyhuta', 'Dublin'])`

b. Gondolkodhatunk fordítva is: a ciklusban a fővárosok lista elemeit járjuk be, és megnézzük, hogy melyik elem található meg az átadott listában.
A `['Bécs', 'Bécs']` listát vizsgálva melyik megoldás ír válaszul egyet, melyik kettőt?
c. Írjuk meg a főprogramot! A lista elemeit a `lista_neve.append('új érték')` függvénnyel bővíthetjük. Mielőtt a listát átadnánk az eljárásnak, írjuk ki a lista elemeit egy utasítással! A lista elemeit vesszővel (és szóközzel) válasszuk el! Segít a `join()` függvény.

5. Írjunk olyan eljárást, amely egy paraméterként kapott szóról eldönti, hogy magánhangzóval vagy mássalhangzóval kezdődik, és a döntését képernyőre írja!
   a. Hogyan tudjuk egy változóban tárolt szó első betűjét megkapni?
   b. Meg kell vizsgálnunk, hogy a megkapott első betű benne van-e egy listában. A mássalhangzókat vagy a magánhangzókat érdemes listába gyűjtenünk?




```
eljárás magánhangzó_mássalhangzó(szó)
    magánhangzók = [’a’, ’á’, ’e’, ’é’, ’i’, … ]
    ha szó[0] eleme magánhangzók-nak:
        ki: „magánhangzó”
    különben:
        ki: „mássalhangzó”
eljárás vége
```

```
1. def magánhangzó_mássalhangzó(szó):
2.     magánhangzók = ['a', 'á', 'e', 'é', 'i', 'í', 'o', 'ó',
3.                     'ö', 'ő', 'u', 'ú', 'ü', 'ű']
4.     if szó[0] in magánhangzók:
5.         print(szó, 'magánhangzóval kezdődik.')
6.     else:
7.         print(szó, 'mássalhangzóval kezdődik.')
```

Akármilyen zárójelen belül bármikor kezdhetünk új sort, de írhatjuk egy sorba is.

Függvények a gyakorlatban

A függvények abban különböznek az eljárásoktól, hogy van visszatérési értékük, amit a `return` utasítást követően adunk meg a függvény belsejében. A függvények is írhatnak a képernyőre, de nem szokás ezzel a módszerrel élni. Az iparban a függvényt nagyon gyakran nem ugyanaz a fejlesztő írja, aki majd a programjában használja. Azt szeretjük, ha a függvény fekete doboz: a függvényt használó fejlesztőnek nem kell ismernie a függvény belső működését. Nem tudja, mi történik belül a „dobozban”, csak átadja a függvénynek a feldolgozandó értékeket, a függvény meg visszaadja az eredményt.

Eddig is rengeteg fekete dobozként működő függvényt használtunk – valójában függvény minden olyan „beépített” utasításunk, aminek a neve után zárójel van. Függvény a `len()`, az `int()` és az `str()`, a `join()`, a `random.randint()`. Említettük, hogy a Python az eljárásokra visszatérési érték nélküli függvényként tekint, így például a `print()`-re is, ami annak alapján, amit tanultunk, inkább egy eljárás.

Bár mi egyszerre vagyunk a függvényeink fejlesztői és használói, mégis a fekete doboz elvét szem előtt tartva írjuk meg függvényeinket. Természetesen a függvény a fejlesztése alatt, tesztelési-hibakeresési célból írhat a képernyőre, de a kész függvény már ne tegyen ilyet.

Feladatok

1. Vegyük elő a múltkori akós programunkat, és írjuk át úgy, hogy eljárás helyett függvény legyen benne! Ne felejtkezzünk meg a főprogram módosításáról sem!

```
függvény akóba_vált(liter):
    vissza liter/58,6
függvény vége
ki: „999 liter az”, akóba_vált(999), „akó.”
```

```
1. def akóba_vált(liter):
2.     return liter/58.6
3.
4. print('999 liter az', akóba_vált(999), 'akó.')
```

2. Írjunk olyan függvényt, amely a paraméterként kapott egyjegyű pozitív számot betűkkel leírva adja vissza! Ötlet: a számok neveit írjuk listába, és használjuk a `számok_nevei[szám]` hivatkozást.

```
1. def betűkkel(szám):
2.     számok_nevei = ['nulla', 'egy', 'kettő', 'három', 'négy', …]
3.     return számok_nevei[szám]
4.
5. for szám in range(10):
6.     print(szám, betűkkel(szám))
```

3. Írjunk olyan függvényt, ami a paraméterként kapott szóhoz a megfelelő határozott névelőt adja vissza! A függvény nagyon hasonlít ahhoz az eljárásunkhoz, amelyikkel eldöntöttük, hogy az átadott szó elején magán- vagy mássalhangzó áll.


```
függvény névelő(szó)
    magánhangzók = ['a', 'á', 'e', 'é', 'i', … ]
    ha szó[0] eleme magánhangzók-nak:
        vissza: "az"
    különben:
        vissza: "a"
függvény vége
```

```
1. def névelő(szó):
2.     magánhangzók = ['a', 'á', 'e', 'é', 'i', 'í', 'o', 'ó',
3.                     'ö', 'ő', 'u', 'ú', 'ü', 'ű']
4.     if szó[0] in magánhangzók:
5.         return 'az'
6.     else:
7.         return 'a'
```

Kihívást jelentő feladat: Hogyan tudjuk ezt a függvényt a számok neveit visszaadóval együtt arra használni, hogy a számok elé is a megfelelő névelőt tegye a programunk?

4. Írjunk olyan függvényt, amelynek visszatérési értéke a paraméterként kapott szó hangrendjét adja meg! Szükségünk lesz egy listára a magas, és egy másikra a mély magánhangzókkal. Érdemes lehet logikai változó értékének beállításával jelezni, hogy találtunk-e magas, illetve mély magánhangzót, majd a két változó értéke alapján meghozni a döntést.

```
függvény hangrend(szó)
    mély_magánhangzók = [„a”, „á”, „o”, „ó”, „u”, „ú”]
    magas_magánhangzók = [„e”, „é”, „i”, „í”, „ö”, … ]
    volt_mély := hamis
    volt_magas := hamis
    ciklus szó minden betű-jére:
        elágazás
        ha betű eleme mély_magánhangzók-nak:
            volt_mély := igaz
        különbenha betű eleme magas_magánhangzók-nak:
            volt_magas := igaz
        elágazás vége
```

```
    ciklus vége
    ha volt_mély és nem volt_magas:
        vissza „mély”
    különbenha nem volt_mély és volt_magas:
        vissza „magas”
    különbenha volt_mély és volt_magas:
        vissza „vegyes”
    különben:
        vissza „nem volt magánhangzó a szóban”
függvény vége
```

```
1. def hangrend(szó):
2.     mély_magánhangzók = ['a', 'á', 'o', 'ó', 'u', 'ú']
3.     magas_magánhangzók = ['e', 'é', 'i', 'í', 'ö', 'ő', 'ü', 'ű']
4.     volt_mély = False
5.     volt_magas = False
6.     for betű in szó:
7.         if betű in mély_magánhangzók:
8.             volt_mély = True
9.         elif betű in magas_magánhangzók:
10.            volt_magas = True
11.    if volt_mély and not volt_magas:
12.        return('mély')
13.    elif not volt_mély and volt_magas:
14.        return('magas')
15.    elif volt_mély and volt_magas:
16.        return('vegyes')
17.    else:
18.        return('nincs magánhangzó a szóban')
19.
20. szó = input('Írj ide egy szót! ')
21. print(hangrend(szó))
```

Lehetne úgy is írni, hogy If volt_mély == True and volt_magas == False, de így elegánsabb.

A négyféle visszatérési érték négy return utasítást jelent.

Programunk ebben a formában csak kisbetűs szavakkal működik helyesen. Hogyan tudunk segíteni a problémán?
Kihívást jelentő feladat: Hogyan oldható meg a nagybetűs szavak helyes feldolgozása a magánhangzók listájának bővítése nélkül?

5. Írjunk olyan függvényt, amely egy nevet kapva paraméterként visszaadja a névből képzett monogramot!

```
1. def monogram(név):
2.     szóköz_helye = név.index(' ')
3.     return név[0] + '. ' + név[szóköz_helye+1] + '.'
4.
5. név = input('Írj ide egy nevet! ')
6. print(név, 'monogramja:', monogram(név))
```

Az index() függvény megadja a paraméterének első előfordulását. Listáknál is működik.

Kihívást jelentő feladat: A program jelen formájában például Zsákos Bilbó monogramját hibásan adja meg. Oldjuk meg a problémát! Ötlet:

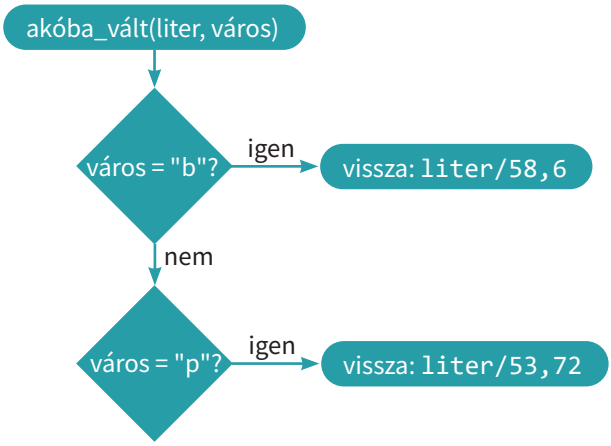
```
if név[0] in ['G', 'L', 'N', 'T'] and név[1] == 'y' …
```

Egy függvénynek több paramétert is átadhatunk. Ilyenkor a paramétereket a függvény neve után álló zárójelben, vesszővel felsorolva adjuk meg. Pont úgy, mint a print() függvény esetében.

```
1. def összeadás(egyik, másik):
2.     return egyik + másik
3.
4. def osztás(osztandó, osztó):
5.     return osztandó/osztó
6.
7. print(összeadás(3, 6))
8. print(összeadás(6, 3))
9.
10. print(osztás(3, 6))
11. print(osztás(6, 3))
```

*A függvények a paraméterek **sorrendjéből** tudják, hogy melyik paramétert melyik változóba tegyék.*

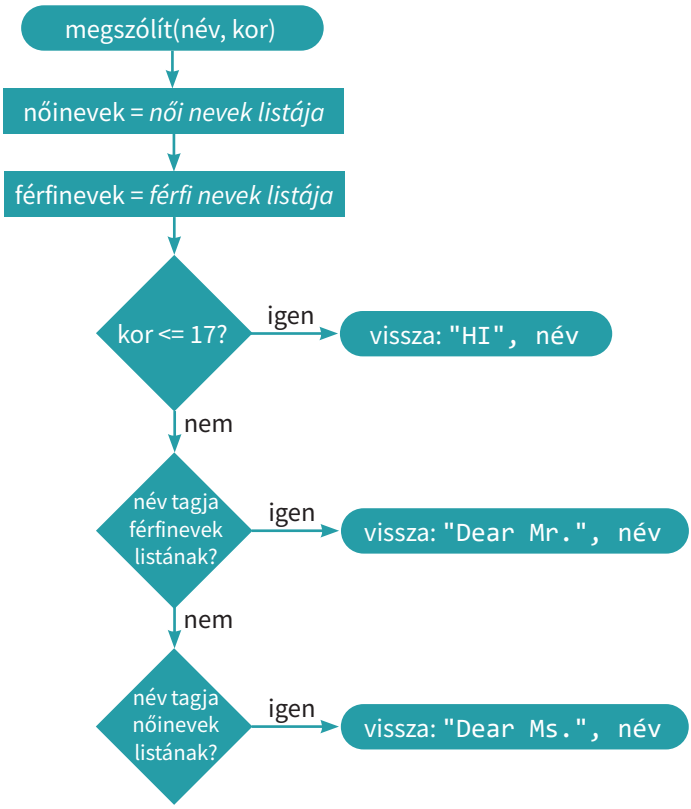
6. Írjuk át az akóátváltós függvényünket úgy, hogy budai és pesti akóba is tudjon váltani! A függvény második paramétere lehet „b” vagy „p”, ez alapján döntse el, hogy mennyivel oszt.




```
függvény akóba_vált(liter, város)
  elágazás
  ha város = „b”:
       vissza: liter/58,6
  különbenha város = „p”:
      vissza: liter/53,72
  elágazás vége
függvény vége
```

```
1. def akóba_vált(liter, város):
2.     if város == 'b':
3.         return liter/58.6
4.     elif város == 'p':
5.         return liter/53.72
6.
7. print('999 liter Budán', akóba_vált(999, 'b'), 'akó.')
8. print('999 liter Pesten', akóba_vált(999, 'p'), 'akó.')
```

7. Írjunk angol megszólítást előállító függvényt! A függvény paramétere egy név és egy évszám. Ha a név Kate, az évszám pedig legfeljebb tizenhét, akkor a megszólítás tegeződős: „Hi Kate”. Ha az évszám nagyobb tizenhétnél, akkor a függvény névlista alapján dönti el, hogy a név férfi vagy női, és a „Dear Mr. Smith” vagy a „Dear Ms. Smith” formát ölti a megszólítás.
   a. Írjuk meg a függvényt úgy, hogy csak egy tagból álló nevet vár!



   b. Nem az első olyan függvényünk ez, amelyik bizonyos esetekben semmit nem ad vissza. Ez kerülendő – adjunk vissza olyan üzenetet, ami elmondja, hogy miért nem adtunk vissza semmi értelmeset!
   c. Módosítsuk úgy a függvényt, hogy két tagból álló nevet kezeljen, a fiatalokat a keresztnevükön, az idősebbeket a vezetéknevükön megszólítva!
   d. Ne csak angolul tudjon a függvényünk! Módosítsuk úgy, hogy harmadik paraméterként egy nyelvet várjon, és a nyelvnek megfelelő megszólítást írjon ki!
   e. Milyen „igazi” alkalmazásban tudunk elképzelni egy ehhez hasonló működésű függvényt?

8. A vogon költészetről az irodalom egyik, mára klasszikussá vált sci-fijéből, a Galaxis útikalauz stopposoknak című műből szerzett tudomást az emberiség. Ha el akarunk benne mélyedni, ám tegyük, de most elég annyit tudni róla, hogy borzasztó. Mi is hasonlóan borzasztó verseket állítunk elő a következő programunkkal.
   A versek sorai mindig egy jelzőből, egy tárgyból, egy állítmányból, valamint mondat végi írásjelből állnak („Nyekergő ablakokat dobálunk!”, „Gömbölyű kútkávát eszünk?!”)

a. Írjunk meg egy olyan, `verssor()` nevű, paraméter nélküli függvényt, amely egy ilyen verssort ad vissza, a mondat négy részét négy listából véletlenszerűen válogatva!
b. Ezt a függvényt hívja egy másik, `versszak()` nevű függvény. A `versszak()` paramétere egy szám, ami megadja, hogy hány sorból álljon a versszak. Ez a függvény a teljes versszakot adja vissza.
c. Az utolsó függvényünk neve: `vers()`. Az első paramétere a versszakok számát megadó szám, a második a verssorok száma versszakonként, visszatérési értéke a teljes vers.
d. A verseink túlcsorduló érzelmekről tesznek tanúbizonyságot, legalábbis az írásjelek alapján. Hogyan tudjuk nagyon egyszerűen elérni, hogy a mondatok végére gyakrabban kerüljön pont?
e. Kihívást jelentő feladat: Írjuk át úgy a `vers()` függvényt, hogy paraméterként mindössze egy listát várjon! Ha a lista [2, 2, 4], akkor olyan verset adjon vissza, amelyik három versszakból áll, és a versszakok rendre 2, 2 és 4 sor hosszúak!

```
1. import random
2.
3. def verssor():
4.     jelzők = ['Nyekergő', 'Gömbölyű', 'Piros', 'Vidám', 'Iszonytató']
5.     tárgyak = ['amőbát', 'ablakokat', 'sárgolyót', 'kútkávát']
6.     állítmányok = ['dobálunk', 'eszünk', 'álmodunk', 'éneklünk']
7.     írásjelek = ['.', '?', '!', '?!']
8.     return random.choice(jelzők) + ' ' + random.choice(tárgyak) + ' ' \
9.         + random.choice(állítmányok) + random.choice(írásjelek)
10.
11. def versszak(sorok_száma):
12.     vsz = ''
13.     for _ in range(sorok_száma):
14.         vsz += verssor() + '\n'
15.     return vsz
16.
17. def egyszerű_vers(versszakok_száma, verssorok_száma):
18.     vers = ''
19.     for _ in range(versszakok_száma):
20.         vers += versszak(verssorok_száma) + '\n'
21.     return vers
22.
23. def jóféle_vers(sorok_számainak_listája):
24.     vers=''
25.     for sorok_száma_a_versszakban in sorok_számainak_listája:
26.         vers += versszak(sorok_száma_a_versszakban) + '\n'
27.     return vers
28.
29. print(egyszerű_vers(3,4))
30. print('---------------')
31. print(jóféle_vers([2,2,4]))
```

A visszaper jelzi, hogy még nincs vége a sornak. Te a gépeden írhatod egy sorba a 8–9. sort. (Ilyenkor nem kell visszaper.)

Variációk típusalgoritmusokra 1.
Történetek a taxisról meg a rókáról

Mik azok a típusalgoritmusok?

Egészen egyszerűen olyan algoritmusok, amelyekkel a programírás során felmerülő problémák egyszerűen megoldhatók. Más néven programozási tételeknek is nevezik őket. Könyvünkben hat egyszerűbb ilyen algoritmussal ismerkedünk meg behatóan, és ezek közül néggyel már találkoztunk is.

A sorozatszámítás – összegzés és átlagolás

1. Egy listában tároljuk, hogy egy taxis egy nap során hány piculát keresett az egyes fuvarjaival. Mennyi pénze lett összesen?

```
bevételek = [1,5,2,3,4]
összes := 0
ciklus bevételek minden bevétel-ére:
    összes := összes + bevétel
ciklus vége
ki: összes
```

```
1. bevételek = [1,5,2,3,4]
2.
3. összes = 0
4. for bevétel in bevételek:
5.     összes += bevétel #vagy: összes = összes + bevétel
6. print('Napi bevétel:', összes, 'picula.')
```

Persze van egyszerűbb megoldás is:

```
1. összes = sum(bevételek)
2. print('Napi bevétel:', összes, 'picula.')
```

Csakhogy ez a rövid megoldás nem minden esetben használható. Például mire megyünk vele a következő feladatban?

2. 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. Hány kiló libát ehet meg a róka?
Az előző feladat algoritmusát felhasználva írjuk meg azt az algoritmust, amelyik ezt a problémát oldja meg!

```
1. libák = [1,5,2,3,4]
2.
3. rókának_jut = 0
4. for liba in libák:
5.     if liba <= 3:
6.         rókának_jut += liba
7. print('A rókának', rókának_jut, 'kilónyi liba marad.')
```

3. Átlagosan hány piculát keres a taxisunk egy fuvarral?
Az előző taxis feladat algoritmusán csak annyit kell módosítanunk, hogy az összeget elosztjuk a lista elemszámával. Ha az első, hosszabb megoldásból indulunk ki, akkor kell még egy változó, ami a ciklus előtt nulla, és a ciklus minden ismétlődésekor eggyel növeljük az értékét. Ez két új sor beszúrása az algoritmusba. Módosítsuk az algoritmust!
A kész kód:

```
1. bevételek = [1,5,2,3,4]
2.
3. összes = 0
4. darab = 0
5. for bevétel in bevételek:
6.     összes += bevétel
7.     darab += 1
8. print('Az átlagos bevétel:', összes/darab, 'picula.')
```

Persze megy ez is egyszerűbben:

```
3. összes = sum(bevételek)
4. darab = len(bevételek)
5. print('Az átlagos bevétel:', összes/darab, 'picula.')
```

Mondjunk olyan átlagolási feladatot, amikor a rövid megoldás nem használható!

4. Átlagosan hány kilósak a rókának maradt libák?

```
1. libák = [1,5,2,3,4]
2.
3. rókának_jut = 0
4. darab = 0
5. for liba in libák:
6.     if liba  <= 3:
7.         rókának_jut += liba
8.         darab += 1
9. print('A róka libái átlagosan', rókának_jut/darab, 'kilósak.')
```

Eldöntés

Megnézzük, hogy van-e adott tulajdonságú elem a listánkban.

5. Volt-e a taxisnak ma ötpiculás bevételű fuvara?
Ezúttal az a feladatunk, hogy addig vizsgálgatjuk ciklussal az értékeket, amíg meg nem találjuk az első olyat, amelyik eleget tesz a feltételnek – ami a mi esetünkben öt.

```
bevételek := [1,5,2,3,4]
vanilyen := hamis
ciklus bevételek minden bevétel-ére:
    ha bevétel = 5:
        vanilyen := igaz
ciklus vége
ha vanilyen:
    ki: „van ilyen”
```

Az algoritmusunk megoldja a problémát, de picit pazarló, nem takarékoskodik a gép erőforrásaival. Végignézi ugyanis az összes értéket a listában, még azt követően is, hogy talált már a feltételnek megfelelőt. Persze ezúttal ez nem gond, de mi van, ha a taxis összes bevétele ott van a listában, mondjuk az előző harminc évről? Szerencsére erre is van megoldás:

```
1. bevételek = [1,5,2,3,4]
2.
3. vanilyen = False
4. for bevétel in bevételek:
5.     if bevétel == 5:
6.         vanilyen = True
7.         break
8. if vanilyen:
9.     print('Van ötpiculás bevétel.')
```

A break arra jó, hogy kiléphessünk a ciklusból, mielőtt az összes elemet bejártuk volna. A többi listalemet úgyis fölösleges volna megvizsgálnunk.

Ahogy már megszoktuk, itt az egyszerűbb megoldás:

```
3. if 5 in bevételek:
4.     print('Van ötpiculás bevétel.')
```

Gondoljunk ki olyan feladatot, ami nem oldható meg a rövid megoldással!

6. Előfordult-e olyan, hogy a róka legalább háromkilós libát lopott?

```
1. libák = [1,5,2,3,4]
2.
3. vanilyen = False
4. for liba in libák:
5.     if liba >= 3:
6.         vanilyen = True
7.         break
8. if vanilyen:
9.     print('Van legalább egy háromkilós liba.')
```

Előfordult-e olyan, hogy a róka kisebb libát hozott, mint az előző napon?
Ebben a feladatban mindenképp index szerint kell bejárnunk a listát, hiszen az aktuális elemet mindig az előzőhöz kell hasonlítani.

```
 1.libák = [1,5,2,3,4]
 2.
 3.vanilyen = False
 4.for index in range(len(libák)):
 5. :   if index > 0:
 6.         if libák[index] < libák[index-1]:
 7.             vanilyen = True
 8.             break
 9.if vanilyen:
10.    print('Volt, amikor kisebb libát lopott az előző napinál.')
```

len(libák) = 5
range(5): 0,1,2,3,4
Az index 0-tól 4-ig fut.

Az első (nulladik sorszámú) elemet még nem vizsgáljuk.

Kiválasztás

Az eldöntéshez képest az a különbség, hogy tudjuk, hogy van adott tulajdonságú elem (ötpiculás bevétel, háromkilósnál nagyobb liba) a listában. De hol van ez az elem? Hányas sorszámú helyen áll?

7. A taxis bevételei közül hányadik volt az ötpiculás?
   Index szerint bejárva a listát végignézzük az egyes elemeket, és megjegyezzük a keresett – és immáron megtalált – elem indexét.

```
bevételek := [1,5,2,3,4]
holvan := speciális_érték
ciklus index := 0-tól bevételek_elemszáma -1 -ig:
    ha bevétel[index] = 5:
        holvan := index
ciklus vége
ki: holvan
```

Észrevehetjük, hogy ez az algoritmus nagyon hasonló az eldöntésnél tárgyalt utolsó esethez – hasonlítsuk össze a két programkódot! Azt is megemlítjük, hogy ez az algoritmus a lehetséges jó válaszok közül mindig csak egyet talál meg. A mondatszerű leírásban az úgynevezett számlálós ciklus szerepel. Az `index` változó értékét változtatja 0 és 4 között. A Python nyelvben ilyen ciklus nincs – helyette bejárósat használunk, a már ismert módon, azaz a bejárandó elemeket a `range()` függvénnyel előállítva. Az index változó ezeket számokat kapja értékül.

```
1. bevételek = [1,5,2,3,4]
2.
3. holvan = None
4. for index in range(len(bevételek)):
5.     if bevételek[index] == 5:
6.     holvan = index
7.     break
8. print('Az ötpiculás fuvar sorszáma:', holvan + 1)
```

A köznyelvben nem nullával kezdjük a számozást – ezért növeljük eggyel a kiírt értéket.

Az egyszerűbb megoldáshoz a lista adattípus index() függvényét használjuk:

```
3.  print('Az ötpiculás fuvar sorszáma:', bevételek.index(5) + 1)
```

8. Megszoktuk már, hogy ilyenkor mindig jön a róka a libáival, és elrontja a jó kis egyszerű megoldás fölött érzett megkönnyebbülésünket. A helyzet most is ez.
   Hányadik napon sikerült a rókának először legalább háromkilós libát lopnia?

```
1. libák = [1,5,2,3,4]
2.
3. holvan = None
4. for index in range(len(libák)):
5.     if libák[index] >= 3:
6.         holvan = index
7.         break
8. print('Az első nagy libát a(z) ', index+1,
9.         '. napon fogta a róka.', sep='')
```

Ezt a két sort írhatjuk egybe is. Ezúttal nem használunk visszapert a sor végén: zárójelen belül nem kell.

Variációk típusalgoritmusokra 2.
Újabb történetek a taxisról meg a rókáról

Keresés

A keresés algoritmusa nem más, mint az eldöntés és a kiválasztás egybeépítése. Az eldöntésnél az a kérdés, hogy van-e olyan elem a listában, amit keresünk, a kiválasztásnál pedig az, hogy melyik ez az elem. Itt mindkét kérdésre válaszolunk.

1. Keressük azt a fuvart, amikor a taxis először keresett öt piculát. Az eldöntés és a kiválasztás algoritmusának ismeretében alkossuk meg az algoritmust, majd kódoljuk.

```
bevételek := [1,5,2,3,4]
vanilyen := hamis
holvan := speciális_érték
ciklus index := 0-tól bevételek_elemszáma -1 -ig:
    ha bevétel[index] = 5:
        vanilyen := igaz
        holvan := index
    elágazás vége
ciklus vége
ha vanilyen:
    ki: holvan
különben:
    ki: „nincs ilyen”
```

Kódolva:

```
1. bevételek = [1,5,2,3,4]
2.
3. vanilyen = False
4. holvan = None
5. for index in range(len(bevételek)):
6.     if bevételek[index] == 5:
7.         vanilyen = True
8.         holvan = index
9.         break
10. if vanilyen:
11.     print('Az ötpiculás fuvar sorszáma:', holvan + 1)
12. else:
13.     print('Nincs ötpiculás fuvar.')
```

Ha találunk megfelelő elemet, mind a két változó értékét beállítjuk.

Ha megszakítjuk a keresést, akkor az első, ha végigmegyünk az összes elemen, akkor pedig az utolsó megfelelő értéket írjuk ki.

Természetesen ezúttal is létezik egyszerűbb megoldás – és ez is a két előző „egyszerűbb megoldás” összeépítése.

```
3. if 5 in bevételek:
4.     print('Az ötpiculás fuvar sorszáma:', bevételek.index(5) + 1)
5. else:
6.     print('Nincs ötpiculás fuvar.')
```

2. Melyik a róka első legalább háromkilós libája?
A rövid megoldás szokás szerint nem használható. A hosszabbat azonban egyetlen karakterben kell *lényegileg* módosítanunk, és máris megoldja ezt a problémát.

Megszámolás

A megszámolás algoritmusával azt derítjük ki, hogy adott tulajdonságú elemből mennyit találunk a listánkban. A lista bejárását végző ciklus előtt létrehozunk egy számláló szerepű változót, és a nulla értéket adjuk neki. Minden egyes találatnál növeljük a számláló értékét.

3. Megszoktuk, hogy a taxis mindig az egyszerű eset: az ő esetében nem csak néhány elem számít a vizsgálatba, mint a rókánál. Ha egy lista *minden* elemét meg kell számolnunk, akkor ugyebár a lista elemszámára, azaz hosszára vagyunk kíváncsiak. Ez pedig a már elég jól ismert `len(bevételek)` utasítással kiderül.
Úgyhogy kivételesen az egyszerű és rövid megoldással kezdjük a sort, és már ezzel is olyan kérdésre válaszolunk, amelyben egy konkrét értéket számolunk meg.
Hány ötpiculás bevétele volt a taxisnak?

```
1. bevételek = [1,5,2,3,4]
2.
3. ennyi = bevételek.count(5)
4. print('Összesen',  ennyi, 'ötpiculás fuvar volt.')
```

Gondoljunk ki olyan feladatot, amelyikre a fenti megoldás nem használható!

4. Talán még emlékszünk rá, hogy a farkas a három kilónál nagyobb libákat veszi el a rókától. Hány libát tarthat meg a róka? Adjunk algoritmust a feladat megoldására, majd készítsük el belőle a kódolt programot!

```
libák := [1,5,2,3,4]
számláló := 0
ciklus libák minden liba-jára:
    ha liba <= 3:
        számláló := számláló + 1
ciklus vége
ki: számláló
```

Kódolva:

```
1. libák = [1,5,2,3,4]
2.
3. számláló = 0
4. for liba in libák:
5.     if liba <= 3:
6.         számláló += 1
7. print(számláló, 'libája marad a rókának.')
```

Csak akkor számoljuk a soron következő libát, ha elég kicsi.

Maximum- és minimumkiválasztás

A kiválasztás tétele arról szól, hogy tudjuk, hogy van adott tulajdonságú elem a listában, de melyik az? Most is tudjuk, hogy van legnagyobb és legkisebb elem, de melyek azok? A maximum és a minimum kiválasztása végezhető együtt is, de azért mi először külön-külön tesszük ezt meg.

5. Melyik volt ma a taxis legjobb fuvarja? Írjuk meg a problémát megoldó algoritmust, majd kódoljuk programmá! Ötlet: vezessünk be egy változót, aminek az értéke a feladatban elképzelhető legkisebb eredmény. Nézzük végig egyesével a listánk elemeit, és ha találunk a változóban tároltnál nagyobb értéket, akkor cseréljük erre a változó tartalmát.

```
bevételek := [1,5,2,3,4]
legtöbb := 0
ciklus bevételek minden bevétel-ére
    ha bevétel > legtöbb:
        legtöbb := bevétel
ciklus vége
```

```
1. bevételek = [1,5,2,3,4]
2.
3. legtöbb = 0
4. for bevétel in bevételek:
5.     if bevétel > legtöbb:
6.         legtöbb = bevétel
7. print('A legjobb fuvar', legtöbb, 'piculát ért.')
```

Ha a mostani bevétel nagyobb, mint az eddigi legnagyobb, akkor mostantól ez a legnagyobb.

Természetesen van egyszerűbb forma:

```
4. print('A legjobb fuvar', max(bevételek), 'piculát ért.')
5. print('A legrosszabb csak', min(bevételek), 'piculát hozott.')
```

Természetesen az egyszerű forma nem mindig elég. Gondoljunk ki olyan feladatot, ahol kevésnek bizonyul!

6. Mekkora a legkisebb liba, amit a farkas elvesz a rókától?

```
1. libák = [1,5,2,3,4]
2.
3. farkas_legkisebb_libája = 100
4. for liba in libák:
5.     if liba >= 4:
6.         if liba < farkas_legkisebb_libája:
7.             farkas_legkisebb_libája = liba
8. print('A farkasnak jutó legkisebb liba',
9.         farkas_legkisebb_libája, 'kg.')
```

A farkas csak a négykilós vagy nehezebb libákkal törődik.

Ha a mostani liba kisebb, mint az eddigi legkisebb, akkor mostantól ez a legkisebb.

Az, hogy a farkas legkisebb libáját tároló változó kezdőértékéül a 100-at választottuk, bátor döntés. Ha ugyanis a róka libáit tartalmazó listában csupa 1-2-3 kilós madár kerül, a végén a programunk azt hazudja majd, hogy a farkas legkisebb libája egy százkilós, strucc méretű liba.

7. Kihívást jelentő feladat: Módosítsuk úgy az előző programunkat, hogy biztosan helyesen adja meg a farkasnak jutó legkisebb liba tömegét!

Fejtörők

Melyik típusalgoritmus való a következő feladatok megoldására? Elég az egyszerűbb forma, vagy kell-e a részletes? Elég érték szerint bejárnunk a listát, vagy index szerint kell?

8. Listában tároljuk, hogy egy londoni pincér mekkora összegeket helyezett el a pénztárcájában az elmúlt órában. Amikor kivett a tárcából pénzt, azt negatív számmal jeleztük. A pincérünk elég ügyetlen és slampos, így soha nem kap borravalót. `[3, 8, 10, 19.35, -6, 5.1, 9, 20]`
   a. Volt-e olyan, hogy a pincér vásárolt valamit, vagy mindig csak neki fizettek?
   b. Ha az óra elején üres a pénztárcája, mennyi van benne az óra végén?
   c. Hány alkalommal kapott biztosan pennyt is, nem csak fontot?
   d. Hány pennyt kapott összesen (feltételezve, hogy csak azt fizették pennyben, amit nem lehet fontban)?
   e. Hány esetben kapott legalább öt fontot?
   f. Mi állt a legnagyobb számlán, amit fizettek a pincérnél?
   g. Ha az óra elején már volt 8 font 23 penny a tárcájában, mennyi pénz volt benne az óra végén?
   h. Hányadik vendég fizetett 9 fontot?
   i. Ha volt olyan vendég, aki tíz fontnál többet fizetett, akkor mondjuk meg, hogy hányadik vendég volt az első ilyen!
   j. Ha volt olyan vendég, aki tíz fontnál többet fizetett, akkor mondjuk meg, hogy hányadik vendég volt az utolsó ilyen!
   k. Volt-e olyan vendég, akinek módjában állt csupa ötfontossal kiegyenlíteni a számlát?
   l. Ha a főnöke minden vendég után fél fontot ad pincérünknek fizetésül, mekkora bevétellel zárta az órát?

9. Tudjuk, hogy a karakterláncok sokszor listaként viselkednek, hosszuk például megállapítható a `len()` függvénnyel. A `'betűK'.isupper()` függvény megmondja, hogy a string minden karaktere nagybetű-e. Ha van egy gyümölcsneveket tartalmazó listánk, akkor az `if 'ló' in gyümölcsök` utasítással megtudjuk, hogy a ló gyümölcs-e.

   A következő feladatok egy mondat szavaiból képzett listára vonatkoznak.

   ```
   ['Én', 'elmentem', 'a', 'vásárba', 'fél', 'pénzen.']
   ```

   a. Hány szóból áll a mondat?
   b. Hány betűs a legrövidebb szó?
   c. Van-e a mondatban olyan szó, ami írásjellel végződik?
   d. Hány névelő van a mondatban, illetve a szavait tartalmazó listában?
   e. Hányadik szó a „fél”?
   f. Van-e a mondatban nagy kezdőbetűs szó, és ha igen, akkor hol?

Variációk típusalgoritmusokra 3.

Feladatok

Ebben a leckében az előző lecke fejtörőit megoldó programokat készítjük el.

Az első feladatcsoportban listában tároljuk, hogy egy londoni pincér mekkora összegeket helyezett el a pénztárcájában az elmúlt órában. Amikor kivett a tárcából pénzt, azt negatív számmal jeleztük. A pincérünk elég ügyetlen és slampos, így soha nem kap borravalót.

A lista: `[3, 8, 10, 19.35, -6, 5.1, 9, 20]`

1. Volt-e olyan, hogy a pincér vásárolt valamit, vagy mindig csak neki fizettek?
   A megoldáshoz az eldöntés típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem egy konkrét érték előfordulását vizsgáljuk.

```
1. bevételek = [3, 8, 10, 19.35, -6, 5.1, 9, 20]
2.
3. vásárolt_a_pincér = False
4.
5. for bevétel in bevételek:
6.     if bevétel < 0:
7.         vásárolt_a_pincér = True
8.         break
9.
10. if vásárolt_a_pincér:
11.     print('Volt, hogy a pincér vásárolt is.')
12. else:
13.     print('Csak a pincérnek fizettek.')
```

Az első két sor a pincéres programjainkban megegyezik, a további példamegoldások a 3. sortól kezdődnek.

2. Ha az óra elején üres a pincér pénztárcája, mennyi pénz van benne az óra végén?
   A megoldáshoz az összegzés típusalgoritmusát használjuk – elegendő az egyszerű forma, mert minden értéket figyelembe kell vennünk.

```
3.  print('A pénztárcában', sum(bevételek), 'font van.')
```

3. Hány alkalommal kapott pennyt is, nem csak fontot?
   A megoldáshoz a megszámolás típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem egy konkrét érték előfordulásait számoljuk meg.
   Akkor kap pennyt is, ha pozitív törtszám a lista vizsgált eleme. Azt, hogy egy szám törtszám-e, kétféleképp is eldönthetjük – mindkét módszer kigondolható az *Elemi adattípusok és elágazások* című lecke utolsó példájában leírtak alapján.

```
 4. ennyiszer_kapott_pennyt = 0
 5.
 6. for bevétel in bevételek:
 7.     if bevétel > 0 and bevétel % 1 != 0:
 8.         ennyiszer_kapott_pennyt += 1
 9.
10. print('A pincér', ennyiszer_kapott_pennyt,
11.         'alkalommal kapott pennyt.')
```

Ha fizetett is pennyvel, azt nem számoljuk, csak azt, amikor kapott.

4. Hány pennyt kapott összesen a pincér?
A megoldáshoz az összegzés típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem minden értéket összegzünk.
Egy angol font száz pennyt ér.

```
 3. ennyi_pennyt_kapott_fontban = 0
 4.
 5. for bevétel in bevételek:
 6.     if bevétel > 0 and bevétel % 1 != 0:
 7.         ennyi_pennyt_kapott_fontban += bevétel % 1
 8.
 9. print('A pincér', ennyi_pennyt_kapott_fontban * 100,
10.         'pennyt kapott.')
```

Csak a bevétel tört részével foglalkozunk.

Alighanem látni fogjuk, hogy a pennyk száma nem egész szám, sok nullát követően egy pici érték még lesz a végén. Ez a probléma a törtszámok kettes számrendszerben való tárolásából adódik. Több módszer is kínálkozik, hogy úrrá legyünk a helyzeten, például:

- az eredmény kerekítése – a `round()` függvénnyel végezhető;
- az eredmény egésszé alakítása – az `int()` függvény használatával;
- a törtszámok nagyobb pontossággal való tárolása – a decimal modul dolga.

5. Hány esetben kapott legalább öt fontot a pincér?
A megoldáshoz a megszámolás típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem egy konkrét érték előfordulásait számoljuk meg.

```
 3. ennyiszer_kapott_legalább_5_fontot = 0
 4.
 5. for bevétel in bevételek:
 6.     if bevétel >= 5:
 7.         ennyiszer_kapott_legalább_5_fontot += bevétel
 8.
 9. print('A pincér',
10.         ennyiszer_kapott_legalább_5_fontot,
11.         'alkalommal kapott legalább 5 fontot.')
```

Írjuk át úgy a programot, hogy ne mindig öt fonthoz viszonyítson – kérdezzük meg a számot a felhasználótól!

6. Mi állt a legnagyobb számlán, amit fizettek a pincérnél?
A megoldáshoz a maximumkiválasztás típusalgoritmusát használjuk – elegendő az egyszerű forma, mert minden értéket figyelembe kell vennünk.

```
3.  print('A legnagyobb számlán', max(bevételek), 'állt.')
```

7. Ha az óra elején már volt 8 font 23 penny a tárcájában, mennyi pénz volt benne az óra végén?
A megoldáshoz az összegzés típusalgoritmusát használjuk – elegendő az egyszerű forma, mert minden értéket figyelembe kell vennünk. A második feladathoz képest annyi a különbség, hogy 8,23-at hozzá kell adnunk a `sum()` függvény eredményéhez.
Írjuk át úgy a programot, hogy egy függvény végezze el a számítást! A függvény paramétere a `bevételek` lista. Ha elkészültünk, módosítsuk úgy a programot, hogy a kezdeti összeget is a felhasználótól kérdezze meg, és ezt is adjuk át paraméterként a számítást végző függvénynek!

A float() függvénnyel alakítjuk az input()-tól kapott szöveget törtszámmá.

A round() függvénnyel kerekítünk két tizedesre.

```
4. def kiszámol(bevételek_listája, kezdeti_pénz):
5.     végső_pénz = sum(bevételek_listája) + kezdeti_pénz
6.     return végső_pénz
7.
8. eleve_a_tárcában = float(input('Mennyi pénz \
9. van a pincérnél az óra elején? '))
10.
11. print('A pénztárcában',
12.         round(kiszámol(bevételek, eleve_a_tárcában), 2),
13.         'font van.')
```

8. Hányadik vendég fizetett 9 fontot?
A megoldáshoz a keresés típusalgoritmusát használjuk – elegendő az egyszerű forma, mert minden értéket figyelembe kell vennünk, és egy konkrét értéket keresünk.

```
3.  print('A ', bevételek.index(9)+1,
4.          '. vendég fizetett 9 fontot.', sep='')
```

9. Ha volt olyan vendég, aki tíz fontnál többet fizetett, akkor mondjuk meg, hogy hányadik vendég volt az első ilyen!
A megoldáshoz a keresés típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem egy konkrét érték előfordulását keressük. A listát kénytelenek leszünk index szerint bejárni, mert szükségünk lesz a megtalált érték listában elfoglalt helyére is. Ha nagyon ódzkodunk az index szerinti bejárástól, akkor az is jó megoldás, hogy megtaláljuk az első tíznél nagyobb értéket, majd a 8. feladatnál is látható `index()` függvénnyel megkérdezzük, hogy hányadik helyen áll.

```
van_több_mint_10 = False
hol_van = 0
for index in range(len(bevételek)):
    if bevételek[index] > 10:
        van_több_mint_10 = True
        hol_van = index
        break

if van_több_mint_10:
    print('Az első tíz fontnál többet fizető vendég a(z) ',
            hol_van+1, '. vendég.', sep='')
```

10. Ha volt olyan vendég, aki tíz fontnál többet fizetett, akkor mondjuk meg, hogy hányadik vendég volt az utolsó ilyen!
A megoldás nagyon hasonló az előző feladatéhoz. Az utolsó vendég fellelésére két lehetséges módszer:

- Kihagyjuk a `break` utasítást – ilyenkor végiglépdel a ciklusunk az összes listaelemen, és az utolsó tíznél nagyobb elem pozíciója marad a `hol_van` változóban. Ez hosszú listáknál pocséklás, azaz a megoldás működik, de nem hatékony.
- A bejárandó indexeket előállító `range()` függvényt így paraméterezzük: `range(len(bevételek)-1, -1, -1)`. A `len()` 8-at ad vissza, ezért az első –1 használatával kivonunk belőle egyet, hogy a lista utolsó értékétől, azaz héttől járjuk be a listát. A második –1 az első olyan értéket adja meg, amit a `range()` már nem ad vissza – azaz 0-ig tart a bejárás. A harmadik –1 azt mondja meg, hogy egyesével akarunk haladni visszafelé. Ez a korrekt megoldás. Sok más nyelven ezt egy visszafelé haladó számlálós ciklussal valósítanánk meg – de Pythonban nincs ilyen.

11. Írjuk ki, ha volt olyan vendég, akinek módjában állt csupa ötfontossal kiegyenlíteni a számlát!
A megoldáshoz az eldöntés típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem egy konkrét érték előfordulását vizsgáljuk. Csak akkor kell kiírnunk valamit, ha találunk megfelelő értéket, ezért picit egyszerűsítünk a megoldáson.

```
for bevétel in bevételek:
    if bevétel % 5 == 0:
        print('Volt olyan vendég, aki tudott csupa ötössel fizetni.')
        break
```

12. Ha a főnöke minden vendég után fél fontot ad pincérünknek fizetésül, mekkora bevétellel zárta az órát?
A megoldáshoz a megszámolás típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem minden értéket akarunk megszámolni, és nem egy konkrét érték előfordulásainak számát.

```
3. vendégek = 0
4.
5. for bevétel in bevételek:
6.     if bevétel > 0:
7.         vendégek += 1
8.
9. print('A pincér', round(vendégek/2, 2 ), 'font fizetést kap.')
```

A második feladatcsoport feladatai a következő, egy mondat szavait tartalmazó listára vonatkoznak:

```
['Én', 'elmentem', 'a', 'vásárba', 'fél', 'pénzen.']
```

13. Hány szóból áll a mondat?
A megoldáshoz a megszámolás típusalgoritmusát használjuk – elegendő az egyszerű forma, mert minden értéket figyelembe kell vennünk.

```
1. mondat = ['Én', 'elmentem', 'a', 'vásárba', 'fél', 'pénzen.']
2.
3. print('A mondat', len(mondat), 'szóból áll.')
```

Az első két sor a további programjainkban megegyezik, a következő példamegoldások a 3. sortól kezdődnek.

14. Hány betűs a legrövidebb szó?
A megoldáshoz a minimumkiválasztás típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem a lista értékei, hanem az azokból képzett számok között keresünk.

```
3. legrövidebb_szó_hossza = 1000
4.
5. for szó in mondat:
6.     if len(szó) < legrövidebb_szó_hossza:
7.         legrövidebb_szó_hossza = len(szó)
8.
9. print('A legrövidebb szó', legrövidebb_szó_hossza, 'karakteres.')
```

Olyan kezdeti értéket választunk, aminél biztosan van kisebb.

Ha a mostani szó rövidebb, mint az eddigi legrövidebb, akkor mostantól ez a legrövidebb.

Kis gond van a programunkkal: a szó hosszába beleszámítja az írásjeleket is. Ezen most nem fogunk segíteni.

15. Írjuk ki, ha van a mondatban olyan szó, ami után mondatvégi írásjel áll!
A megoldáshoz az eldöntés típusalgoritmusát használjuk, ráadásul kétszer is. A fő eldöntéshez a részletes algoritmus kell, mert nem közvetlenül a lista értékeit vizsgáljuk. Az egyes szavak végének vizsgálatakor viszont jól jön az egyszerű forma. Vegyük észre, hogy az írásjeleket tartalmazó karakterláncban kereshetünk lista módjára – bár használhatnánk listát itt is.

```
3.  írásjelek = '.?!'
4.
5.  for szó in mondat:
6.      if szó[-1] in írásjelek:
7.          print('Van olyan szó, ami után írásjel áll.')
```

A –1. index az utolsó karaktert jelenti.

16. Hány névelő van a mondatban?
A megoldás nagyon hasonló az előző feladatéhoz – ezúttal nem csak az utolsó karaktert hasonlítjuk, hanem az egész szót, és az írásjelek helyett a névelők listájára van szükség.

17. Hányadik szó a „fél”?
A megoldáshoz a keresés típusalgoritmusát használjuk – elegendő az egyszerű forma, mert minden értéket figyelembe kell vennünk, és teljes szót keresünk.

```
3.  print('A mondatban a "fél" szó a', mondat.index('fél')+1,'. helyen áll.', sep='')
```

18. Van-e a mondatban nagy kezdőbetűs szó, és ha igen, akkor hol?
A megoldáshoz a keresés típusalgoritmusát használjuk – a részletes algoritmus kell, mert nem a lista értékei, hanem az azokból képzett értékek figyelésével hozunk döntést. A listát index szerint járjuk be, mert azt is tudni akarjuk, hogy hányadik szó kezdődik nagy kezdőbetűvel. Az `isupper()` függvénnyel vizsgáljuk, hogy egy adott betű nagybetű-e.

```
3.  van_nagy_kezdőbetűs = False
4.  hol_van = None
5.  for index in range(len(mondat)):
6.      if mondat[index][0].isupper():
7.          van_nagy_kezdőbetűs = True
8.          hol_van = index
9.
10. if van_nagy_kezdőbetűs:
11.     print('A(z) ', hol_van+1,
12.             '. szó kezdődik nagybetűvel.', sep='')
```

A mondat indexedik szavának nulladik sorszámú karaktere.

A fenti kód sem hatékony. Miért? Hogyan oldható meg, hogy amikor már találtunk nagybetűs szót, hagyjuk is abba a keresést?
Módosítsuk úgy a programot, hogy a keresést végző rész kerüljön függvénybe! A függvény paramétere a mondat szavait tartalmazó lista, a függvény visszatérési értéke pedig egy lista, a nagybetűs szavak indexeivel.

Listákat tartalmazó listák – kétdimenziós adatszerkezet

Mik azok a kétdimenziós adatszerkezetek?

Az egyszerű listák egydimenziós adatszerkezetek – azaz csak hosszuk van, mint egy szakasznak a geometriában. Azonban a listákban elhelyezhetünk olyan elemeket is, amelyek saját maguk is listák. Így lesz az adatszerkezet kétdimenziós: van „szélessége” és „magassága.”

A következő lista egy vonósnégyes tagjainak jelenléti íve. Az ív egy hét munkanapjain mutatja a jelenlétet, ahol 1 szerepel benne, ott jelen volt a zenész, ahol 0, ott nem. Egy-egy „kis lista” egy zenész jelenlétét mutatja be, a „nagy” lista pedig az egész zenekarét.

```
1.  ív = [[1, 1, 1, 1, 1],
2.       [1, 1, 1, 1, 0],
3.       [1, 1, 0, 0, 0],
4.       [0, 1, 1, 1, 1]]
5.
```

egyik hegedűs

másik hegedűs

brácsás

csellós

Figyeljük meg, hogy a kis listák között vessző van – hiszen most a kis listák a nagy lista elemei, márpedig egy lista elemeit vesszővel soroljuk fel. Látjuk, hogy a csellós hétfőn hiányzik, péntekre pedig kiről a második hegedűs és a brácsás. Hogy lesz így előadás szombaton?!

A kétdimenziós adatszerkezetek használata nagyon gyakori – pont úgy, ahogy a táblázatoké a valóságban.

Listákat tartalmazó listák bejárása

Amikor ezt a kétdimenziós listát bejárjuk, a ciklusváltozóba mindig egy-egy vonós egész heti jelenléti íve, azaz egy kis lista kerül. Próbáljuk ki!

```
6.  for vonós in ív:
7.      print(vonós)
```

Ha az egyes vonósok jelenléti ívét tartalmazó kis listákat is be akarjuk járni, akkor egymásba ágyazott ciklusokat kell írnunk:

```
6.  for vonós in ív:
7.      for nap in vonós:
8.          if nap == 1:
9.              print('itt', ' ', sep='', end='')
10.         else:
11.             print('otthon', ' ', sep='', end='')
12.     print()
```

Ez a print() az egyes zenészek sorai végén új sort kezd.

Típusalgoritmusok a kétdimenziós adatszerkezetekben

Az alábbi példákban a zenészes listánkkal dolgozunk. A példakódok mindig a hatodik sortól indulnak, mert az első négy sor (meg utána egy üres sor) csak a jelenléti ív adatait tartalmazza – a korábban ismertetett módon.

1. A zenészek a közeli kifőzdében szoktak ebédelni. Ismerik őket, úgyhogy hét közben csak felírják a fogyasztott adagok számát, és péntekenként fizetik az egész heti számlát. Hány adagot fizetnek ezen a pénteken?
   Az első megoldásban egyszerűen megszámoljuk, hogy hány egyes van a „táblázatban”:

```
adagok = 0
for vonós in ív:
    for nap in vonós:
        if nap == 1:
            adagok += 1  #vagy adagok = adagok + 1
print('Összesen', adagok, 'adagot kell fizetni pénteken.')
```

   A második megoldásban egy vonós heti fogyasztását a `sum()` függvénnyel adjuk meg. Ez a függvény összeadja az átadott listában vagy más bejárható objektumban lévő számokat: `sum([4,3,2]) = 9; sum(range(4)) = 6.`

```
adagok = 0
for vonós in ív:
    adagok += sum(vonós)
print('Összesen', adagok, 'adagot kell fizetni pénteken.')
```

   összegzés egyszerű formája egy sorban

   Az előző megoldásban kihasználtuk, hogy történetesen egyesekkel jelezték a jelenlétet. Ha például a ’jelen volt’ karaktersorozattal jelezték volna, a megoldásunk fabatkát sem érne. Ilyenkor nem összeadnunk, hanem megszámolnunk kell, amire a megszámolás típusalgoritmusának egyszerű formája, a `count()` függvény alkalmas. Ezt a lista után írjuk, egy ponttal a listához kötve. A függvény egyébiránt karakterláncokkal, azaz stringekkel is működik: `'rendetteremtettem'.count('e') = 6.`

```
adagok = 0
for vonós in ív:
    adagok += vonós.count(1)
print('Összesen', adagok, 'adagot kell fizetni pénteken.')
```

   megszámolás egyszerű formája egy sorban

   A Python nem volna Python, ha nem lehetne egyetlen sorban is elegánsan megoldani ezt a feladatot – akit érdekel, keressen a *list comprehension* (azaz listaértelmezés) és a *flattening* (azaz lapítás) kifejezésre az interneten!

2. Melyik zenész volt a legtöbbet jelen a héten? Melyik a legkevesebbet?
   Erre a kérdésre jobb híján a zenész sorszámával válaszolunk. A sok lehetséges megoldásban két nagyobb csoportot különítünk el.
   Az elsőben egyetlen ciklus lefuttatásával kapjuk meg a megoldást. A ciklus lényegében egy maximumkiválasztás. Ha arra gondolunk, hogy az összes érték között keresünk, és az értékeket nem manipuláljuk – nem a négyzetük vagy köbgyökük között keressük a legnagyob-

bat –, akár eszünkbe juthatna az algoritmus egyszerűbb formáját is használni. Aztán eszünkbe jut, hogy ez azért nem lesz jó, mert *maguk az értékek nem állnak rendelkezésre azonnal feldolgozható formában*. Így a ciklusban először előállítjuk az értékeket, és párhuzamosan vizsgáljuk, hogy az épp soron következő zenész többször volt-e jelen, mint az eddigi legtöbbet jelen lévő.

```
 6. sorszám = None
 7. legnagyobb = 0
 8.
 9. for index in range(len(ív)):
10.     vonós_jelenléte = sum(ív[index])
11.     if vonós_jelenléte > legnagyobb:
12.         legnagyobb = vonós_jelenléte
13.         sorszám = index
14. print('A(z) ', sorszám+1, '. vonós volt a legtöbbet jelen.', sep='')
```

összegzés egyszerű formája a maximumkiválasztás kellős közepén

A második megoldáscsoport úgy fog hozzá a probléma megoldásához, hogy egy ciklussal először előállítja azt a listát, amiben keresgélni kell. Ha ezt a listát előállítottuk, megkeressük a legnagyobb értéket (maximumkiválasztás), majd megtudjuk az érték pozícióját (kiválasztás).

```
 6. vonósok_jelenlétei = []
 7.
 8. for vonós in ív:
 9.     vonós_jelenléte = sum(vonós)
10.     vonósok_jelenlétei.append(vonós_jelenléte)
11.
12. sorszám = vonósok_jelenlétei.index(max(vonósok_jelenlétei))
13.
14. print('A(z) ', sorszám+1, '. vonós volt a legtöbbet jelen.', sep='')
```

Ez a ciklus tölti fel az új listát az append() függvényt használva.

maximumkiválasztás

keresés

3. Volt-e olyan zenész, aki mindig jelen volt?
A feladat az előzőnél egyszerűbb: nem kell maximumot keresni, tudjuk, hogy azt kell eldöntenünk, hogy volt-e valaki ötször.

4. Írjuk ki, ha volt olyan nap, amikor mindenki jelen volt a próbán!
A feladat nehézsége, hogy ezúttal nem egy vonóst, hanem egy napot vizsgálunk. Úgy is mondhatjuk, hogy nem egy sor értékeit összegezzük, hanem egy oszlopét.

```
 6. for nap_indexe in range(5):
 7.     e_napon_ennyien_voltak = 0
 8.     for vonós in ív:
 9.         e_napon_ennyien_voltak += vonós[nap_indexe]
10.     if e_napon_ennyien_voltak == 4:
11.         print('A(z) ', nap_indexe+1,
12.               '. napon mindenki jelen volt.', sep='')
```

Összegzés: 0 vagy 1 hozzáadása. Minden „napon” lefut.

eldöntés

Objektumok adatai kétdimenziós listákban

Programozásórán még nem hangsúlyoztuk, de épp ideje szokni azt a gondolatot, hogy a programjaink *objektum*okkal – árucikkekkel, tanulókkal, órarendekkel, menetrendekkel, könyvekkel és még ki tudja mivel – dolgoznak. Végső soron objektumok szerepeltek az előző, jelenléti íves példában is, de ezúttal jelenlétük nyilvánvalóbb lesz. Egy osztály tanulóinak adatait tároljuk egy 2D-listában. Egy-egy kis lista egy-egy objektumról, azaz egy-egy diákról szól. A kis lista elemei, azaz az objektumainkról tárolt tulajdonságok: név, nem, kor és e-mail-cím. Íme:

```
1.  osztály = [['Noémi', 'l', 15, 'noemi@hipp.hopp'],
2.             ['Dezső', 'f', 17, 'dezso2@nyikk.nyekk'],
3.             ['Gizella', 'l', 16, 'gizi@pikk.pakk'],
4.             ['Edömér', 'f', 16, 'edo@itt.ott']]
5.
```

A példakódok a már ismert módszer szerint a hatodik sornál kezdődnek.

1. Írjuk ki az osztály névsorát! Írjuk ki minden név mellé az e-mail-címet is!

```
6.  for tag in osztály:
7.      print(tag[0], ": ", tag[-1], sep='')
```

4. Mennyi az osztály átlagéletkora?

```
6.  összeg = 0
7.  for tag in osztály:
8.      összeg += tag[2]
9.
10. print('Az osztály átlagéletkora', összeg/len(osztály), 'év.')
```

9. A lányok vannak többen, vagy a fiúk?

```
6.  lányok = 0
7.  for tag in osztály:
8.      if tag[1] == 'l':          <-- kis L betű
9.          lányok += 1
10.
11. if lányok > len(osztály) - lányok:
12.     print('A lányok vannak többen.')
13. elif lányok < len(osztály) - lányok:
14.     print('A fiúk vannak többen.')
15. else:
16.     print('A fiúk pont annyian vannak, mint a lányok.')
```

10. Kérjünk be egy nevet a felhasználótól, és válaszoljunk a megfelelő ember e-mail-címével! Ötlet: tároljuk a nevet egy változóban, majd járjuk be a listát, és ahol `tag[0]` a megadott név, ott írjuk ki `tag[-1]`-et! Kell-e `break` a kiírást követően?

Objektumok szótárban

A szótár adattípus

Eddig egyetlen összetett adattípusunk van, mégpedig a lista. (Illetve van még egy, az úgynevezett range típus, amit a `range()` függvénnyel állítunk elő, de önmagában szinte semmire nem használjuk.) Ebben a leckében megismerkedünk a szótár adattípussal. Előrebocsátjuk, hogy a szótár adattípus nem *szükséges* abban az értelemben, hogy minden, amire a szótár típus használható, megoldható listák használatával is, de a szótárak használata olyan sok esetben teszi kényelmesebbé a munkánkat, hogy kifejezetten *érdemes* megismerkednünk vele.

A megismerkedéshez írjunk egy egyszerű magyar–angol szótárat. (Senki ne gondolja azonban, hogy a szótár adattípussal csak a köznapi értelemben vett szótárat lehet készíteni!) A szótárunkban tárolunk néhány szót, és a programunknak képesnek kell lenni arra, hogy amíg a felhasználó üres bemenetet nem ad, addig megkeresi neki a magyar szó angol megfelelőjét. A szótárunkat első közelítésben még a szótár adattípus használata nélkül, pusztán a lista adattípus használatával valósítjuk meg.

```
1.  def szótáraz(magyar_szó):
2.      szótár = [['nagy', 'big'], ['pici', 'tiny'],
3.                ['én', 'I'], ['foci', 'soccer'],
4.                ['ott', 'there'], ['szarvas', 'deer'],
5.                ['soha', 'never']]
6.
7.      for szó in szótár:
8.          if szó[0] == magyar_szó:
9.              return szó[1]
10.
11.     return 'Nincs ilyen szó a szótárban.'
12.
13.
14. kérdezett_szó = None
15.
16. while kérdezett_szó != '':
17.     kérdezett_szó = input('Melyik magyar szóra vagy kíváncsi? ')
18.     if kérdezett_szó != '':
19.         print(kérdezett_szó, 'keresésének eredménye:',
20.               szótáraz(kérdezett_szó))
```

Minden szópár egy kis lista. A gépeden írhatod egy sorba.

keresés

Itt kezdődik a főprogram.

Itt hívjuk a függvényt.

Ez a szokásos kérdezős ciklusunk – addig kérdez, amíg üres bemenetet nem kap.

A szótárazás pont olyan feladat, hogy már érdemes kiszerveznünk önálló függvénybe, hogy a kódunk könnyen olvasható maradjon. A feladatot egy ciklus végzi, amiben a keresés algoritmusát alkalmasan átalakítottuk. Nem tartjuk nyilván, hogy volt-e találat, hiszen, ha volt, a 9. sorban lévő `return` úgyis megszakítja a függvény futását, és visszatér a találattal.

A második közelítésben már bevetjük a szótár adattípust, és menet közben meg is ismerkedünk vele. Aki lépésről lépésre hasonlítja össze az új programot az előzővel, annak szólunk, hogy itt nem lesz függvényünk. A szavakat tároló adatszerkezetünk, azaz egy szótár nevű és szótár típusú szerkezet így néz ki:

```
1. szótár = {'nagy': 'big', 'pici': 'tiny',
2.           'én': 'I', 'foci': 'soccer',
3.           'ott': 'there', 'szarvas': 'deer',
4.           'soha': 'never'}
```

Figyeljük meg, hogy nincsenek „kis listák". **A szótár egy megfeleltetési adattípus, amiben kulcs-érték párokat helyezünk el.** Egy ilyen kulcs-érték pár a `'nagy':'big'`. A *nagy* a kulcs, a neki megfelelő érték a *big*. Az ilyen kulcs-érték párosok felsorolása a szótár, amit – megkülönböztetendő a listától – kapcsos zárójelekben adunk meg.

A szótárak és a kétdimenziós listák használata közötti leglényegesebb különbség, hogy egy szó kikeresése ezúttal csak ennyi: `print(szótár['nagy'])`. Írjuk csak be ötödik sornak, és próbáljuk ki! Nincs ciklus, nem használjuk a keresés típusalgoritmusát. Ha már látjuk, hogy ez milyen kényelmes, valósítsuk meg a szótárprogramunkat a szótár használatával!

Az 5. sor üres, ezért nem írjuk ide.

```
6. kérdezett_szó = None
7.
8. while kérdezett_szó != '':
9.     kérdezett_szó = input('Melyik magyar szóra vagy kíváncsi? ')
10.    if kérdezett_szó != '':
11.        print(kérdezett_szó, 'keresésének eredménye:',
12.              szótár[kérdezett_szó])
```

Ha alaposan teszteltük a programunkat, bizonyára feltűnt, hogy nem törődtünk azzal az esettel, amikor a szótárban nincs meg a keresett érték. Ráadásul a programunk el is halálozik, ha nem létező értéket kerestetünk vele. A helyzetet megoldja a szótár adattípus `get()` függvénye. Cseréljük le a 12. sort erre!

```
12.              szótár.get(kérdezett_szó, 'Nincs ilyen szó.'))
```

A szótáraknak azonban csak a kulcsai között tudunk ilyen egyszerűen keresni. Ha a magyar–angol szótárunkat angol–magyarrá alakítanánk, a listás megoldásnál egyszerűen `szó[0]` helyett `szó[1]`-et írunk, és fordítva. A szótáras megoldásnál viszont ilyenkor már tényleg be kell járnunk a szótárat. A bejárás alapesetben a kulcsokat teszi a ciklusváltozóba: `for kulcs in szótár`. Ha az érték is kell, akkor a ciklus belsejében használhatjuk a `szótár[kulcs]` formát, de pythonosabb, ha egyszerre kérjük el a kulcsot és az értéket is a `for kulcs, érték in szótár.items()` sorral. Próbáljuk ki a szótárunk összes értékének kiírását mindkét módszerrel!

Megjegyezzük még, hogy

- egy szótár kulcsai között sosem lehet két azonos.
- a szótár kulcsait visszakapjuk a `szótár.keys()`, az értékeket a `szótár.values()` függvénnyel – és ezek is bejárhatók, valamint használható velük az `in` operátor vagy a `sum()` függvény.
- a szótárak értékei maguk is lehetnek listák vagy szótárak, és a szótárak is tehetők listába.
- létező szótárba a `szótár['új kulcs'] = 'új érték'` utasítással vehető fel új érték, a szükségtelenné vált kulcs-érték pár törlésére pedig a `del szótár['törlendő kulcs']` utasítás való.

Osztálytagok a szótárban

A szótárakat nagyon gyakran használjuk arra, hogy egy-egy objektum tulajdonságait tároljuk egy szótárban. Az előző lecke utolsó feladatában, ahol az osztálytagok adatait – tulajdonságait – tároltuk, a tárolást listával oldottuk meg. Ott, ha például valakinek a korára voltunk kíváncsiak, azt nekünk kellett tudni, hogy a kis lista 0. elemeit figyelve keresünk, majd a 2. elemet visszaadva kapjuk meg a választ. Ebben változtat nagyot a szótár, hiszen ha ügyesen alkotjuk meg, elég majd ennyit mondanunk: `print(osztály['Dezső'][kor])`. Ez kényelmesebb megoldás, de nem ez a lényeg, hanem az, hogy első ránézésre látjuk, hogy mit csinál a kód. Manapság ugyanis egy szoftverfejlesztő sokkal gyakrabban foglalkozik egy régi, esetleg nem is őáltala írt kód karbantartásával, javítgatásával, mint új programok írásával.

Ahogy a kétdimenziós listánkat szótárrá alakítjuk, talán ez lesz az első megálló (az e-mail-címek tárolásáról lemondunk, mert a könyvben túl hosszúak lennének a sorok, és áttekinthetetlen lenne a szerkezet):

```
1. osztály = [{'név': 'Noémi', 'nem': 'l', 'kor': 15},
2.            {'név': 'Dezső', 'nem': 'f', 'kor': 17},
3.            {'név': 'Gizella', 'nem': 'l', 'kor': 16},
4.            {'név': 'Edömér', 'nem': 'f', 'kor': 16}]
```

Nos, ez még csak egy szótárakat tartalmazó lista. Ha Dezső korára vagyunk kíváncsiak, a programunk így néz ki:

```
6. for tag in osztály:
7.     if tag['név'] == 'Dezső':
8.         print(tag['kor'])
```

Azaz tudunk már név szerint hivatkozni az egyes tulajdonságokra, és ez jó. Továbbra is be kell járnunk a listát, ez nem jó. Legyünk bátrabbak, és legyen az egész lista egy olyan szótár, amiben a nevek a kulcsok, az értékek pedig az egyes osztálytagok többi tulajdonságát tároló kisebb szótárak! (Figyelem, ezzel a megoldással lehetetlenné tesszük két azonos nevű osztálytag tárolását!)

```
1. osztály = {'Noémi': {'nem': 'l', 'kor': 15},
2.            'Dezső': {'nem': 'f', 'kor': 17},
3.            'Gizella': {'nem': 'l', 'kor': 16},
4.            'Edömér': {'nem': 'f', 'kor': 16}}
5.
6. print(osztály['Dezső']['kor'])
```

Ezért küzdöttünk!

Ha elővesszük az osztályhoz kapcsolódó többi feladatot az előző leckéből, látjuk, hogy mindegyiket meg tudjuk szótárral is oldani.

1. Mennyi az osztály átlagéletkora?

```
6. összeg = 0
7. for tag in osztály:
8.     összeg += osztály[tag]['kor']
9.
10. print('Az osztály átlagéletkora', összeg/len(osztály), 'év.')
```

Ránézésre tudjuk, hogy mit csinál a kód! ☺

2. A lányok vannak többen, vagy a fiúk?
   A megoldást nem mutatjuk meg teljes egészében – az előző lecke 9. feladatának kiírást végző része változtatás nélkül használható itt is.

```
6. lányok = 0
7. for tag in osztály:
8.     if osztály[tag]['nem'] == 'l':
9.         lányok += 1
```

kis L betű

A vonósnégyes és a szótár

Elgondolkodunk még azon, hogy miként érdemes megadnunk az előző lecke vonósnégyesének jelenléti ívét. Lehet például egy olyan szótárunk, amelyikben a kulcsok az egyes zenészek, az értékek pedig a jelenléti listák:

```
1. ív = {'egyik hegedűs': [1, 1, 1, 1, 1],
2.       'másik hegedűs': [1, 1, 1, 1, 0],
3.       'brácsás': [1, 1, 0, 0, 0],
4.       'csellós': [0, 1, 1, 1, 1]}
```

Így már valóban tudunk válaszolni arra, hogy *ki* hiányzott a legtöbbet (és nem csak sorszámot adunk), és mi annak a napnak a *neve*, amikor senki nem hiányzott a próbáról. Természetesen ezúttal is lehetőség van az egyes zenészek listáit újabb szótárakká alakítani. Ekképp látszanának az egyes napok nevei.

Sok esetben van tehát értelme szótárakat használnunk, de sok az olyan eset is, amikor a szótárak használata csak bonyolítaná a programunkat. Az okos fejlesztő végiggondolja a lehetőségeit, mielőtt nekikezd egy program megvalósításának.

Kétdimenziós listák és szótárak a gyakorlatban

Híres szakemberek az adatszerkezetekről

Fred Brooks, a világ egyik leghíresebb számítógéptudósa mondta még 1975-ben:

„Mutasd meg a folyamatábráidat, és rejtsd el a táblázataidat, és homályban maradok. Mutasd meg a táblázataidat, és rendszerint nem kellenek majd a folyamatábrák – minden nyilvánvaló lesz.” – Mai szemmel talán érdemes a folyamatábra helyére a „kód” szót, a táblázat helyére az „adatszerkezet” szót tennünk, és látjuk, hogy tényleg így van, mind a mai napig.

Linus Torvalds, az első Linux megalkotója és a Linux kernel karbantartását végző, mára hatalmas fejlesztői csapat feje szerint „A rossz programozók aggódnak a kódjuk miatt. A jó programozók az adatszerkezeteikkel és azok kapcsolataival törődnek.”

Eric S. Raymond szerint „Az okos adatszerkezet és a buta kód sokkal jobban működik, mint fordítva.”

Guido van Rossum, a Python megalkotója pedig arra figyelmeztet: „Könnyű olyan hibákat elkövetni, amik csak később jönnek elő, miután rengeteg kódot megírtunk. Az ember ráébred, hogy másik adatszerkezetet kellett volna használni. Kezdhetjük elölről.”

Feladatok

Rendelkezésünkre áll egy bolthálózat négy boltjának ezer forintban kifejezett bevétele az elmúlt hét napból.

130	29	143	133
156	15	222	132
231	210	98	182
112	11	101	121
96	191	184	148
311	14	201	199
231	302	87	187

Írjunk olyan programot, eljárást, függvényt, ami az alábbi kérdésekre válaszol!

1. Melyik boltnak a legnagyobb a bevétele az elmúlt hét napban?
2. Melyik boltnak a legnagyobb a tegnapi bevétele?
3. Melyik boltban legnagyobb a legjobb és a legrosszabb nap közötti különbség?

```
1. bevételek = [[130,156,231,112,96,311,231],
2.              [29,15,210,11,191,14,302],
3.              [143,222,98,101,184,201,87],
4.              [133,132,182,121,148,199,187]]
5.
6. legnagyobb_különbség = 0
7. legnagyobb_különbség_helye = None
8. for index in range(len(bevételek)):
```

```
    különbség  = max(bevételek[index]) - min(bevételek[index])
    if különbség > legnagyobb_különbség:
        legnagyobb_különbség = különbség
        legnagyobb_különbség_helye = index

print('A legnagyobb különbség:', legnagyobb_különbség, 'ami ebben a
boltban volt:', legnagyobb_különbség_helye+1)
```

4. Mennyi volt a bolthálózat bevétele a tegnapi napon?
5. Volt-e olyan nap, amelyiken a bolthálózat bevétele elérte a hatszázezer forintot?

Rendelkezésünkre áll a néhány európai ország ezer főben mért népessége (a régebbi adatok az akkor azon a területen élőkre vonatkoznak).

Ország	Évszám			
	0	1000	1500	2000 (1998-as adat)
Franciaország	5000	6500	15000	58805
Németország	3000	3500	12000	82029
Olaszország	7000	5000	10500	57592
Nagy-Britannia	800	2000	3942	59237

(Forrás: https://en.wikipedia.org/wiki/Demographics_of_Europe, 2020. 09. 17.)

Írjunk olyan programot, eljárást, függvényt, ami az alábbi kérdésekre válaszol!

6. Melyik „országnak” a legnagyobb a népessége időszámításunk kezdetekor?
7. Melyik az az „ország”, ahol találunk olyan évet, amelyben az előző adathoz képest alacsonyabb a népesség?

A megoldást kezdjük egy olyan függvény megírásával, amely eldönti egy átadott számlistáról, hogy van-e benne csökkenés!

```
def volt_e_csökkenés(számok):
    for index in range(1,len(számok)):
        if számok[index] < számok[index-1]:
            return True
    return False


országok = {'Franciaország': [5000, 6500, 15000, 58805],
            'Németország': [3000, 3500, 12000, 82029],
            'Olaszország': [7000, 5000, 10500, 57592],
            'Nagy-Britannia': [800, 2000, 3942, 59237]}

for ország in országok:
    if volt_e_csökkenés(országok[ország]):
        print(ország, '-ban volt ilyen.')
```

A bejárást nem a lista legelején kezdjük.

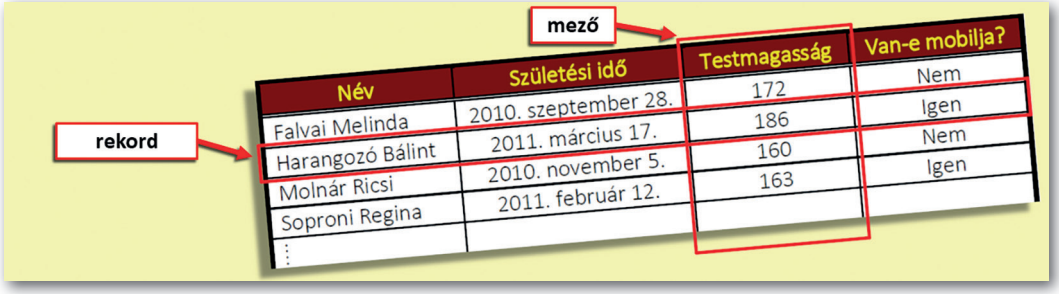
8. Melyik „országnak” hányszorosára nőtt a népessége a megfigyelt két évezred alatt?

Adatbázis-kezelés

Alapfogalmak

Mindenki zsigerileg, ösztönösen meg tudja különböztetni a dolgokat. Persze nem a pillangó és az acélkohó különbségére gondolunk, hiszen az eléggé nyilvánvaló, hanem arra, hogy a hasonló dolgok mi mindenben különböznek. Az eltérő jellemzőket érzékeljük másságként, ez segít a megkülönböztetésükben.

Nézzünk egy példát, hogy ne legyen annyira ködös ez a megfogalmazás! Képzeljünk el egy osztályt, a 10.g-t! Ott van a barna hajú Molnár Ricsi, aki 160 cm magas és 59 kg. Padszomszédja a nyúlánk Harangozó Bálint, aki fekete hajú, 186 cm magas és 58 kg. Ezeket a többé-kevésbé eltérő jellemzőket nevezzük **adatoknak**. Egy-egy tanuló adatainak összességét adatsornak, szép tudományos nevén **rekordnak** hívjuk, a rekordok egyes adatait **mezőnek**, az adatok megnevezését pedig **mezőnévnek**. Akár össze is állíthatnánk egy csinos kis táblázatot néhány adatból:



Ez persze a táblázatnak csak egy része, de ahhoz elegendő, hogy láthassuk, az adataink különböző **típusúak** lehetnek. Egy-egy adat lehet **szöveg**, **szám**, **dátum**, **idő**, **logikai érték** és még egy sor más is eszünkbe juthat, mondjuk a szem színe. Az ilyen táblázatokat **adattáblának** nevezzük. A tábla sorai a rekordok, **oszlopai** a mezők.

Az élet számtalan helyén találkozhatunk ilyen, de akár egész más kinézetű adattáblákkal. Ilyen a buszmegállóba kihelyezett menetrend, a Spotify lejátszási lista számai, a boltban az élelmiszeren az összetétel és kalória tájékoztatója, a naplóba beírt jegyek, a színház havi műsorrendje, de még az éttermi étlap is.

Az egymással logikai kapcsolatban álló adattáblákból **adatbázist** állíthatunk össze, de az akár egyetlen táblából is állhat. A rendszer attól válik adatbázissá, hogy a táblákban tárolt adatok alapján ki lehet nyerni belőle a minket érdeklő információkat. Ez így persze megint elég ködös, jobb, ha nézünk néhány példát.

- Mikor jön a következő busz, ha reggel háromnegyed hét van?
- Mennyibe kerül három liter tej és húsz kifli?
- Mikor indul holnap vonat Budapestről Kecskemétre?
- Melyik edzőcipőből kapható 47-es méretű?

- Mikszáth Kálmánnak mely művei tölthetők le a Magyar Elektronikus Könyvtárból?
- A tatabányai vasútállomás forgalmistája hányszor ázik meg, ha egész nap esik az eső, és minden vonat indításakor ki kell állnia a peronra?
- Milyen határok között mozog ma az ausztrál dollár árfolyama a magyar bankoknál?
- Milyen színekben kaphatók német gyártmányú SUV-ok?

A fenti kérdések szinte sugallják, hogy az egyes adatbázisok adatai megváltozhatnak. Az persze elég nagy káoszt okozhatna, ha bárki módosíthatná az adatokat; mondjuk az áruházban bóklászva észreveszi valaki, hogy kifogyott a hosszú szemű rizs, és törölné azt a sort az áruház adatbázisából. Vannak emberek, akiknek az a feladatuk, hogy naprakésszé tegyék az adatbázisokat, ők módosíthatják az adatok értékét, új sorokat adhatnak a táblákhoz, esetleg törölhetik az elavultakat. A többiek csak megtekinthetik az adatokat, kereshetnek közöttük. Ez az eltérő hozzáférési jogosultság biztosítja rendeltetésszerű működésüket.

Ebben a tanévben még csak online adatbázisokkal fogunk ismerkedni, hogy mindenki jól megérthesse az alapfogalmakat, továbbá rendet vágunk a **nem**, az **és**, továbbá a **vagy** szavak precíz informatikai értelmezésében.

Feladatok

1. Mennyibe kerül a holnapi IC-vonatjegy Szolnokról Békéscsabára, másodosztályon pót- és helyjeggyel?
2. Valamelyik időjárásjelző portálon keressünk rá, hogy a következő hét napban összesen hány milliméter csapadék várható!
3. Egy online számítógépes áruházban nézzünk utána, hogy mennyibe kerül a legolcsóbb vezeték nélküli (wireless) egér!
4. Hová indul a Liszt Ferenc Nemzetközi Repülőtérről a legközelebbi felszálló repülőgép, és mi a járatszáma? Ha a választ egy rekord mezőiként kapjuk, akkor milyen más mezők olvashatók még a járatról?

Vágjunk utat az adatdzsungelben!

Mekkora egy adattábla?

Gondoljunk csak bele, hogy az előző példák közül mondjuk a vasúti vagy a repülőtéri adatbázisnak mennyi járatról kell adatokat tárolni, és járatonként milyen sokat! Érezhető, hogy a járatadatokat tároló tábláknak nagyon sok sorból és oszlopból kell állniuk. Amikor még nem terjedtek el az internetes adatbázisok, vagyis néhány évtizede, a magyar vasúti mentrendet évente több száz oldalas könyvként nyomtatták ki (egészen 2017-ig létezett nyomtatott változat). Ha ebben kellene a Szolnok–Békéscsaba viszonylat kívánt adatait (IC, másodosztály, holnap) megtalálni, az bizony rengeteg keresgéléssel járna, és ezért hosszú ideig tartana.

Persze lehetne azt mondani, hogy töröljünk ki minden más adatot a táblából, és akkor sokkal könnyebb dolgunk lesz. Ez méltánylandó ötlet, bár a táblát összeállítók dühöngenének, hogy hiába dolgoztak az adatbevitellel annyit. Ha ez a kifogás sem tartana vissza az adattörléstől, akkor gondoljunk a következőre! Mihez kezdenénk, ha a mai lecke végén egy másik járat, mondjuk a Székesfehérvár–Veszprém adatairól érdeklődne egy feladat?

Alkalmazzunk szűrőt!

Amikor szüretkor préselik a szőlőt, a kicsorgó gyümölcslé magával sodorhat némi szőlőhéjat és -magot, esetleg a fürtök közé került szennyeződéseket, rovarokat. Ahhoz, hogy ezektől ne romolhasson meg a lé, át szokták szűrni, hogy a szűrőn fennakadjon a felesleg. Sok más folyamatban is alkalmazunk szűrőket a kellő és a fölösleges dolgok elkülönítésére, például bizonyára már mindenki látta, miként szűrik le a megfőtt tésztát. A tészta nem fér át a szűrő lyukain, viszont a főzővíz azokon át el tud távozni.

Szinte hallom a reklamációkat, hogy ez két különböző dolog, hiszen a gyümölcslénél az a szemét, ami fennmarad a szűrőn, a tésztánál azonban a fennmaradó részből lesz sajtos vagy mákos tészta, esetleg spagetti. A méltatlankodásnak van némi alapja, de ha úgy nézzük, hogy szétválasztjuk a szükséges és szükségtelen dolgokat, akkor mindegy, melyik marad fenn a szűrőn, a lényeg a szétválasztás.

A digitális világban az utóbbi módszer működik, **az adatbázisok szűrői** azokat a sorokat mutatják meg, amelyek a szűrési feltételnek vagy feltételeknek megfelelnek, a többit elrejtik a szemünk elől. Hogy a dolog kellően szemléletes legyen, gondoljunk a fejezet első példájára: az iskola tanulóinak adattáblájából kiszűrhetjük a 10.g tanulóit, így máris lesz egy osztálynévsorunk. Itt álljunk meg egy pillanatra! Nem szeretnénk elvarratlan szálakat hagyni.

Rendezés

A 10.g-ben tanulók nevének listája nem lesz automatikusan a hagyományos értelemben vett osztálynévsor, hiszen ahhoz a neveket ábécérendben kellene felsorolni. Az összes adat közül kiválogatottaknál szükség lehet többféle sorrendre.

Például ilyen sorrend lehet:

- neveknél az ábécérend;
- tornasornál a testmagasság;
- orvosi váróban a betegek érkezési időpontja;
- online áruházban a termékek ára;
- lakberendezési áruházban a kész függönyök magassága;
- vonatjáratoknál az indulási idő;
- internetes keresőprogramnál a keresésnek megfelelő oldalakat felkeresők száma;
- településeknél az ott élők lélekszáma.

Az adatbázis-kezelők persze a **rendezésre** is lehetőséget biztosítanak, az online felületeken általában csak konkrét rendezési módok között választhatunk. Ez általában rendben is van így, hiszen ki is szeretné a **találatokat** (a szűrő által kiválogatott rekordokat) mondjuk az árut beszállító életkora vagy az áru és a csomagolás súlyaránya szerint tanulmányozni. De ezek a lehetőségek is arról tanúskodnak, hogy az adatokat lehet rendezni.

Irányított szűrés

Ma már szinte elképzelhetetlen lenne az életünk az online adatbázisok nélkül, hiszen a mindennapjainkban meglehetősen sűrűn fordulunk hozzájuk. A teljesség igénye nélkül lássunk egy elég tág felsorolást! Online adatbázishoz fordulunk, ha

- egy település vagy egy utca helyét keressük a térképen;
- vásárolnánk egy online áruházban;
- szeretnénk megtudni, hogy holnap mikor indul távolsági busz vagy vonat Egerbe;
- szeretnénk mozi- vagy koncertjegyet venni;
- egynémely hivatalos ügyünket szeretnénk elintézni, mondjuk letiltani az elveszett bankkártyánkat;
- képet keresünk a neten;
- klipet keresünk kedvenc videómegosztónkon;
- egy új applikációt szeretnénk feltelepíteni a tabletünkre vagy a mobilunkra;
- ebédet, vacsorát, netán pizzát rendelnénk valahonnan;
- szeretnénk megnézni, hogy milyen jegyeink vannak földrajzból.

Az eddig leírtak azt sugallják – és ha belegondolunk, így is van –, hogy ezekben az adatbázisokban tényleg irdatlan mennyiségű adatot tárolnak. Ha ezeket egyszerre a nyakunkba zúdítanák, szinte lehetetlen lenne kiválasztanunk belőle a számunkra szükséges adatokat. Valami olyasmi lenne, mintha a testvérünk a szülinapja előtt a kezünkbe nyomná a *Háború és béke* meglehetősen vaskos kötetét azzal a megjegyzéssel, hogy aláhúzott benne egy szót, ami arra utal, hogy mit is szeretne ajándékba.

Amikor egy ilyen adatbázist használunk, nem mindig vagyunk tudatában annak, hogy egy nagy táblázatban keresgélünk. Ennek persze az az oka, hogy az oldalon nem mutatják a táblázatrácsot, hisz az adatok elhelyezkedése már tájékoztat, így az emiatt fölöslegessé vált rácsot elrejtik.

Ebben a kásahegyben szinte esélyünk se lenne rátalálni a minket érdeklő adatokra, ha az oldalak üzemeltetői nem kínálnának fel szűrőket. Például a termékcsoportokat a webáruházakban vagy az utazási viszonylatokat a menetrendekben. Ezek persze a tervezőik elképzelései szerint szabályozottak, nem kérhetünk bármilyen szűrést. Néha ez bosszantó, de általában kellő változatosságot biztosítanak, hogy megtalálhassuk a kívánt adatokat. A szűrt adatok mindig valamiféle rendezettséggel jelennek meg a képernyőn, ez lehet például ár szerinti a webáruházakban, indulási idő a menetrendekben stb.

Még a szűrés ellenére is előfordulhat, hogy rengeteg adat közül lennénk kénytelenek válogatni, gondoljuk csak el, hogy például egy elektronikai webáruház kínálatában több száz fajta monitor szerepel. Ilyenkor általában lehetőségünk van további segédszűrők alkalmazására. Például ilyen lehet:

- értékhatár;
- márka;
- alkategória (pl.: tejtermékek → tejek, sajtok…);
- méret;
- szín;
- ügycsoport;
- a keresett zene előadója;
- a keresett kép mérete;
- átszállás nélküli járat a vonatoknál.

Néha arra is van lehetőség, hogy a rendezés szempontját átállítsuk, mondjuk a monitor ára szerintit a monitor képátlójának mérete szerintire. Ezek is gyorsíthatják a keresett adatok közül a számunkra megfelelő kiválasztását.

Ezek a lehetőségek többnyire elégségesek, de néha bosszantó, hogy pont olyan szűrést nem lehet kérni, ami nekünk a legmegfelelőbb lenne. Például a vasúti menetrendben nem lehet csak a délutáni vonatokra keresni. De ennyi kényelmetlenségért biztos kárpótol minket az a kényelem, hogy rengeteg dolgot otthonról intézhetünk. Bizony néhány évtizede még telefonon kellett érdeklődni a vonatok indulásáról, esetleg elzarándokolni a vasútállomásra és ott informálódni, ha nem vettük meg otthonra az adott évi vasúti menetrendet.

Feladatok

1. Keressünk fel két-három elektronikai áruházat, UHD (Ultra High Definition – nagyfelbontású) televíziót vásárolnánk maximum 110 cm-es képátlóval! Írjuk fel áruházanként a kívánt készülékek legmagasabb és legalacsonyabb árát! Jegyezzük fel azt is, hogy hol milyen segédszűrés, rendezés lehetséges!
2. Keressünk képet a Google képkeresőjével, mégpedig légi felvételt a Margit-szigetről. Állítsuk be a következő segédszűrőket: a kép legyen nagyobb 2 MP-nél és álló formátumú! A találatok közül egyet mentsünk le a gépünkre!
3. Látogassunk el egy online élelmiszer-áruházba! Jegyezzük fel, hogy milyen termékkategóriák vannak, nézzük meg és írjuk fel az egyik termékcsoport alkategóriáit is!

Nem nyelvtanóra lesz, vagy mégis?

Mondatok vizsgálata

Találomra kiválasztottunk néhány mondatot J. K. Rowling *Harry Potter és a bölcsek köve* című művéből. Vegyük őket szemügyre!

- *Nappal röpködő baglyok?*
- ***Ron kezéből kiesett a varázspálca.***
- ***Hermione lehorgasztotta a fejét.***
- *Nem tudja eltüntetni, Dumbledore?*
- *Tessék megnézni ezt a kígyót!*
- ***Ronnak elkerekedett a szeme.***
- *Menj ki a postáért, Harry!*
- ***Sötét idők jártak akkoriban, Harry.***
- *Áááááááh!*
- *A betyárját, de szeretnék egy sárkányt!*
- ***Szép, napfényes szombat volt.***

Igen, sikerült választani mind az öt mondatfajtából, láthatunk kijelentő, kérdő, felszólító, óhajtó és felkiáltó mondatot is. Nem véletlenül emeltük ki vastagítással a kijelentő mondatokat, hanem azért, mert a kijelentő mondatoknak van egy olyan tulajdonságuk, ami a másik négy mondatfajtának nincs: **igazságtartalmuk** van. Ez lehet **igaz** vagy **hamis**. Talán érdemes a matematikai elnevezésüket is megismerni, nemcsak azért, mert az informatikusok is ezt a nevet használják, hanem azért, mert ez az elnevezés szemléletesebben utal az igazságtartalomra. Ezentúl a kijelentő mondatokat **állításoknak** hívjuk. Tehát az állítások lehetnek igazak vagy hamisak. Az online adatbázisokban pontosan ilyen állításokat használunk a szűrésre, még akkor is, ha magát az állítást ki sem mondjuk. Például:

- Kedden el kell vonatoznom Budapestről Pécsre.
- Felvágottat szeretnék venni.
- Egy maximum 150 cm szélességű szőnyeg fér még a szobába.
- Kell egy kép a prezentációmba a mamutfenyőkről.
- Pizzát szeretnék rendelni.

Amennyiben ezek alapján szűrünk, csak azokat a rekordokat kapjuk meg, amelyeknél az állítás igaz. Így működnek az adatbázisok szűrői.

Persze ezek mind elég egyszerű állítások voltak, az élet azonban nem mindig ilyen egyszerű. A bonyolultabb esetekhez érezhetően összetettebb állítások tartoznak.

Összetett állítások

Hogyan lehet az egyszerűbbekből összetett állításokat alkotni? Az állításokból némi ügyeskedéssel új állítások készíthetők éppúgy, ahogy a számokból az alapműveletek segítségével más számok állíthatók elő. Persze eszünkbe sem jutna állításokat összeadni, kivonni vagy összeszorozni. A legegyszerűbb összetételekhez elegendő három szó, ezek: a **nem**, a **vagy**, továbbá az **és** szavak. Ezekből épülnek fel a legegyszerűbb **logikai műveletek**. Hogy az összetett állítások szabályszerűségeit könnyen megértsük, a továbbiakban a **„Kedd van”** és az **„Esik az eső”** példamondatokon tanulmányozzuk ezeket.

Tagadás

A számoknál is van egy olyan művelet, amelyhez csak egy szám kell, ez az előjelváltás, így képezhetünk 19-et a –19-ből, vagy –73-at a 73-ból. Az állításoknál ennek megfelelője a **tagadás** művelete, amihez általában a **nem** szót szoktuk felhasználni. A tagadás pont az ellenkezőjére változtatja az állítás igazságtartalmát.

A *Kedd van* állítás tagadása a *Nem kedd van* állítás. Ha mondjuk ma kedd van, akkor az első állítás igaz, a második hamis. Ellenben, ha ma péntek van, akkor az első állítás lesz hamis, viszont a második igaz.

Vigyázat! A *Kedd van* állításnak nem a tagadása a *Péntek van* állítás. Gondoljunk csak bele, hogy például hétfőn mindkettő hamis. Azt is jó tudni, hogy a tagadás nem azonos a hamis szóval, az állítás pedig az igazzal, amint az eddigiekből ez ki is derül. Jegyezzük meg még, hogy a tagadáshoz nem feltétlenül kell a nem szót használni, például a következő mondatpár egymás tagadása: *Ez a sakkbábu fehér. Ez a sakkbábu fekete.*

Műveletek több állítással

Szinte kínálja magát a lehetőség, hogy az egyszerű állításokat kapcsoljuk össze a fentebb már említett kötőszavakkal: *Kedd van, és esik az eső*. Esetleg: *Kedd van, vagy esik az eső.* Az eddigi tapasztalataink meg is adják azokat a szabályokat, ami az összetétel igaz vagy hamis értékét a két eredeti állítás értéke alapján megadja:

Ha két állítást az és szóval kapcsolunk össze, akkor az összetett állítás csak abban az esetben lesz igaz, ha külön-külön mindkét állítás igaz; minden más esetben hamis lesz az összetétel igazságtartalma.

Nézzük csak a példákat!

		Kedd van.	Esik az eső	Kedd van, és esik az eső
Eset	kedden, esőben	igaz	igaz	**igaz**
	kedden, napsütésben	igaz	hamis	**hamis**
	pénteken, esőben	hamis	igaz	**hamis**
	pénteken, napsütésben	hamis	hamis	**hamis**

Hasonlóan járhatunk el a másik, a vagy kötőszóval.

Ha két állítást a vagy szóval kapcsolunk össze, akkor az összetett állítás csak abban az esetben lesz hamis, ha külön-külön mindkét állítás hamis; minden más esetben igaz lesz az összetétel igazságtartalma.

Nézzük csak a példákat!

		Kedd van.	Esik az eső	Kedd van, vagy esik az eső
Eset	kedden, esőben	igaz	igaz	**igaz**
	kedden, napsütésben	igaz	hamis	**igaz**
	pénteken, esőben	hamis	igaz	**igaz**
	pénteken, napsütésben	hamis	hamis	**hamis**

Ezzel a látszólagos kitérővel fontos célunk volt. Az összetettebb szűréseknél e szavakat használjuk a kapcsolat megalkotásakor, azonban a szűrőfeltétel megfogalmazása gyakran félrevezető lehet. Annak érdekében, hogy a későbbiekben ez ne okozzon gondot, lássunk erre is egy tanulságos példát!

Képzeljük el, hogy van egy adattáblánk egy Győrben rendezendő tanári konferenciáról, ahol a tanárok gyakorlati foglalkozásokon vehetnek részt a modern fejlesztési módszerek bemutatóin, utána pedig szekcióüléseken tárgyalják meg a látottakat. Az adattábla az alábbi mezőket tartalmazza a résztvevőkről:

vezetéknév, keresztnév, város, iskola, iskola címe, tanári szak, lakcím

A következő szűrésekre lenne szükségük a szervezőknek:

1) Keresik a matematika és az informatika szakos pedagógusokat.
   Szűrőfeltétel a szak mezőre: *matematika **vagy** informatika*
2) Szeretnék tudni a vidéki résztvevők számát.
   Szűrőfeltétel a város mezőre: ***nem** Budapest.*
3) A Történelemtanárok Egyesülete kérdezi, hogy a fővárosi iskolákból hány történelem szakos résztvevője van a konferenciának.
   Szűrőfeltétel a város mezőre: *Budapest **és***
   a szak mezőre: *történelem*.
4) Az érkezés éjjelén találtak a szálloda előtt egy bőröndöt, amin a névkártyát az eső eláztatta, Az első kettő és a hatodik betű olvasható. A név lehet Kovács és Kocsis is, kié lehet a bőrönd?
   Szűrőfeltétel a vezetéknév mezőre: *Kovács **vagy** Kocsis.*
5) Holnap Abigél és Alex névnap lesz, a szervezők a reggelinél szeretnék felköszönteni a névnapos kollégákat.
   Szűrőfeltétel a keresztnév mezőre: *Abigél **vagy** Alex.*

Talán már ennyiből is látszik, hogy nem szabad automatikusan alkalmazni a feltételekre adott kívánságokban található és, vagy szavakat, és néha jól jön a tagadás is.

Feladatok

1. Mi a hiba a következő okfejtéssel?
   A *Hozzál pizzát!* mondat felszólító, és mégis lehet igazságtartalma. Ugyanis, ha hozol pizzát, akkor igaz lesz, ha nem, akkor pedig hamis.
2. Gondoljuk végig és beszéljük meg, hogy mi történik akkor, ha a tagadást ismét tagadjuk: nem igaz az, hogy nem kedd van.
3. Készítsük el az előzőek mintájára a pontos szűrőfeltételeket az alábbi kívánságokhoz:
   a) Az egyik résztvevő meghallott egy nagyon érdekes okfejtést egy másik résztvevőtől, akinek a névkártyáján az Antal nevet látta, csak azt nem tudja, hogy ez vezeték- vagy keresztnév. Ki lehetett az illető?
   b) A konferencia utolsó napján Bertold és Marietta névnap lesz. Kiket kell üdvözölni ez alkalomból?
   c) Az egyik résztvevő, Deli Anna összebarátkozott egy másik résztvevővel, akinek hamarabb haza kellett utaznia, és tartani szeretné vele a kapcsolatot, de csak annyit tud róla, hogy Budapesten tanít történelmet, a keresztneve pedig Judit.

Vissza a netre!

A Wikipédia

A Wikipédia egy olyan enciklopédia, amelyet az internet felhasználói közösen alkotnak. Ha valaki új dologról szeretne írni benne, módosítaná vagy kijavítaná mások munkáját, estleg hozzáfűzne valamit, akkor bátran nekiállhat. Még csak fiókot sem kell készítenie, hozzá is foghat a szerkesztéshez. Ennek kapcsán a **jogosultságokat** vesszük górcső alá. Természetesen itt is egy adatbázis áll a háttérben, ami a Wikipédia oldalain lévő tartalmakat és azok változásait tárolja.

Ha bárki bármibe belenyúlhatna, már réges-rég káoszba fulladt volna az egész enciklopédia. A Wikipédiának vannak adminisztrátorai, akiknek joguk van törölni, a módosított tartalmat vissza- vagy helyreállíthatják, egyes szerkesztőket hosszabb-rövidebb időre blokkolhatnak, illetve bizonyos lapokra különleges lapvédelmet állíthatnak be. A szerkesztők a forráshivatkozások meglétét is ellenőrzik, ha hiányoznak, akkor elkérik a szerkesztőtől. (Ezzel a munkával az információk hitelességét kívánják biztosítani.)

Ezek a speciális jogosultságok biztosítják, hogy ez a tudástár folyamatosan fejlődhessen, bővülhessen, hiszen bárki hozzáadhat információkat az eddigiekhez, javíthat azokon, vagy frissítheti az időközben elavult, megváltozott adatokat. Az adminisztrátorok folyamatos felügyelete garantálja, hogy valaki tévedéséből vagy szándékos károkozási próbálkozásából ne lehessen helyreállíthatatlan probléma.

A Kréta

Talán nincs is olyan magyar középiskolás, aki ne hallott volna a Kréta rendszerről, hiszen a legtöbbjüknek ez az elektronikus ellenőrzője. Bár a jegyek nyilvántartásán kívül még számos más szolgáltatása is van, de nekünk most ezt is elég lesz átgondolni, hogy jobban megérthessük a jogosultságok rendszerét. Ennek itt is az a feladata, hogy megakadályozza az illetéktelen módosításokat és azt, hogy érzékeny információk szivárogjanak ki és jussanak olyanok kezébe, akikre az nem tartozik.

Egy tanár jogosult arra, hogy a tanulóinak a saját tantárgyából jegyeket írjon be, de a többi tantárgy érdemjegyeit ugyan megnézheti – hiszen a tanárokat köti a titoktartási kötelezettség –, de átjavítani, törölni nem tudja azokat. Az osztályfőnöknek és az iskolavezetésnek a tanárokhoz képest vannak még további többletjogosultságaik, de azok főként a hiányzásokkal, ellenőrzéssel kapcsolatosak.

A tanuló csak megtekintheti a saját jegyeit, a többiekét nem, és a sajátját sem módosíthatja, törölheti. Ez így is van jól, hiszen sokakban feltámadt már a kisördög, hogy törölje vagy kozmetikázza a kapott, nem túl jó jegyét. A rövid távon jónak tűnő csalás ideiglenesen mentesít a vélelmezett retorzióktól, az esetleges szülői büntetéstől, de hosszabb távon sokkal komolyabb következményei lehetnek.

A tanulók szülei is csak a saját csemetéjük osztályzataiba nyerhetnek betekintést.

Közösségi média

A webdizájnerek ugyan elég jól el tudják takarni, de a közösségi média bármelyik formája mögött egy meglehetősen bonyolult adatbázis rejtőzik. Ezeken a csatornákon másokkal kommunikálunk, pl. posztolunk, kommentelünk, tehát közzétesszük, megosztjuk a véleményünket. Az emberek többnyire nem jól reagálnak az övéktől eltérő véleményekre. Az adatbázis és a felhasználók védelme indokolja, hogy többféle biztonsági beállítás védjen a zaklatástól, átveréstől. Azt azért jó tudni, hogy ezek is olyanok, mint a biztonsági öv az autóban, csak akkor védenek, ha használjuk is őket.

Érdemes egyszer rászánni egy kis időt, hogy megnézzük, milyen lehetőségeink vannak. Aki úgy véli, hogy ezeket csak a teljesen amatőrök miatt készítik, az maga is az. Ha végignézzük a biztonsági lehetőségeinket, talán feltámad bennünk némi egészséges bizalmatlanság. Például azon túl, hogy beállítjuk, kik láthatják a posztjainkat, a jövőben az is eszünkbe fog jutni, hogy akkor se adjuk meg bizalmas adatainkat, mondjuk jelszavunkat, ha azt kéri valaki.

Ha az adott médium engedi, kukkantsunk bele a tevékenységnaplóba! Az adatbázis még az évekkel ezelőtti posztjainkat, hozzászólásainkat is tárolja. A felület mögött meg-

húzódó adatbázis méretét sejteti, hogy minden régi dolgot tárol, és mindezt több millió felhasználónál. Bizony ez is az oka annak, hogy néha dühítően lassúnak érezzük az adott médium működését. Nekünk nem érdemes beállnunk az ezen a lassúságon, akadozáson dühöngők táborába, hiszen már ismerjük és értjük az okát.

Az e fejezetben megtárgyalt jogosultságok csak akkor működnek jól, ha mindenki a saját jogosultsági szintjét biztosító felhasználónévvel és jelszóval jelentkezik be az adatbázisba.

Feladatok

1. Indítsuk el a böngészőnket, a címsorába gépeljük be a *https://hu.wikipedia.org* címet! Ez a Wikipédia magyar kezdőlapjára visz. A bal oldalon lévő kínálati listából koppintsunk a *Lap találomra* linkre!
   Lapozzuk végig az oldal felső részén látható *szócikk, olvasás, szerkesztés, laptörténet, vitalap* füleket! Ezekből látható az enciklopédia szerkesztésének sokrétűsége. Zárjuk be a megnyílt lapot minden változtatás nélkül!
2. Keressük fel Magyarország olimpiai bajnokainak listáját és a kémiai elemek listáját tartalmazó Wikipédia-oldalakat!
   a) Van-e lehetőség az oldalakon lévő táblákban szűrésre vagy rendezésre?
   b) Vajon miért?
   c) Ezek közül melyik az az oldal, amely – reményeink szerint – a közeljövőben időről időre frissítésre szorul?
3. Beszéljük meg a tanár irányításával közösen, hogy mi mindenre érdemes ügyelni, ha a közösségi médiát vagy csak úgy általában az internetet használjuk!
   Javasolt szempontok:
   - Mit ne tegyünk soha a világhálón?
   - Milyen oldalakra ne látogassunk el soha?
   - Milyen tartalomra ne kattintsunk rá soha?
   - Mit jelent az, hogy az interneten mindennek nyoma marad? Ez inkább jó vagy rossz?
   - Mire jó az inkognitóablak vagy más néven privát ablak?
   - Milyen adatainkat ne osszuk meg másokkal, főleg ismeretlenekkel?
   - Mennyire kell fenntartással olvasni az információkat a neten? Minden igaz? Minden átverés?
   - Hogyan reagáljunk mások megosztására vagy a saját megosztásaink kommentjeire?
   - Mennyire illik ügyelni mondandónk érthetőségére, a nyelvhelyességre és a helyesírásra?
   - A csupa nagybetűs írás figyelemfelkeltő vagy hibás?

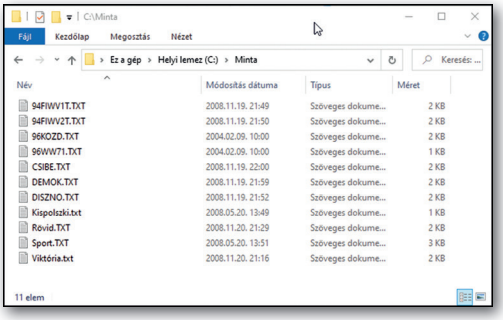
Adatbázis-kezelési fogalmak

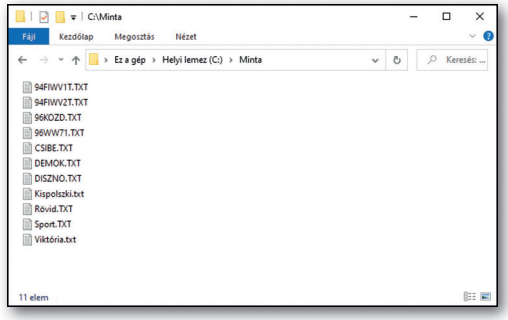
Az adattáblák rendezését és szűrését az előző leckékben részletesen kitárgyaltuk. Sokan talán nem is gondolták, hogy az adatbázisokkal ennyi helyen összetalálkozunk. Ez persze csak a jéghegy csúcsa, az adatbázis-kezeléssel a következő tanévben ismerkedünk majd meg tüzetesebben. Már csak néhány fogalom, metódus jelentésével kell megbarátkoznunk, hogy jövőre ismerősként tekinthessünk rájuk. Jobb ezeket mihamarabb megismerni, simán megérthetőek, így most nem vesznek el a rengeteg, de fontos apróság között.

Nem mindig kell az összes mező

Képzeljük el, hogy egy iskola tanulóinak adattáblájából szeretnénk a 10.g osztálynévsorát megkapni. Ha csak az adott osztály tanulóit látjuk, akkor szükségtelen, de a szűrés velejárója, hogy minden sorban a tanuló nevén kívül látszik a 10.g osztályjelzés is. Pedig elég lenne csak a tanulók neve.

Ezt remekül be tudjuk mutatni a *Fájlkezelő* két különböző nézete, a *Részletek és a Lista* összevetésével. Ha egy jól ismert fájt keresünk, elég a neve is, de ha más információ alapján keressük, akkor nem árt látni minden adatot.





Lekérdezés

Mint már említettük, a nagyobb adatbázisok nemcsak egy, hanem több, egymással összefüggő táblát is tartalmazhatnak. Amikor egy adatbázisból szeretnénk információkhoz jutni, nem biztos, hogy minden adat ugyanabban a táblában van, illetve a táblának, tábláknak nem minden adatára van szükségünk. Ilyenkor összeállíthatunk egy új objektumot, ami kinézetre olyan, mint egy adattábla, de csak az általunk kért mezőket jeleníti meg. Ez a **lekérdezés**. Ezeknél is van lehetőség rendezésre, illetve szűrésre, sőt ezekbe olyan mezőket is bevonhatunk a táblákból, amelyek a végeredményben nem látszanak majd.

Négy fontos dolgot jó még tudni:

- Egyrészt a nagy adatbázisoknál általában lekérdezéseket látunk, azokban szűrünk, rendezünk.
- Másrészt nem árt megjegyezni, hogy lekérdezés többféle is van, ez, amiről eddig beszéltünk, a **választó lekérdezés**, de van még frissítő, törlő, hozzáfűző, kereszttáblás és több más lekérdezésfajta is.
- Harmadrészt a lekérdezéseket megtoldhatjuk számított mezőkkel, amelyek a többi mező adatából képezhetők (például vezeték- és keresztnévből teljes név, vagy az ár és a kedvezmény százaléka alapján a kedvezmény értéke, esetleg az egységár és a

vásárolt mennyiség alapján a vételár, és még sorolhatnánk hosszasan). Ezzel főleg tárolóhelyet és fájlméretet spórolhatunk, mert nem tároljuk azokat az adatokat, amelyek a többi adatból egyszerűen kiszámíthatók, ráadásul az ilyen mezők maguk is automatikusan megváltoznak, ha módosul a számolásban érintett valamelyik adat.

- Negyedrészt a lekérdezések képesek bizonyos adatokat összegezni, átlagolni, adott tulajdonságúakat megszámolni, egy adott mezőben előforduló legkisebb vagy legnagyobb értéket megadni. Az ilyen értékeket kiszámoló lekérdezéseket aggregáló lekérdezéseknek nevezzük.

Jelentés

Igen hasonló a lekérdezésekhez a **jelentés**, ami azzal a többlettel rendelkezik, hogy

- képes a kért adatokat dekoratívan, lapokra bontva, tehát nyomtatásra készen a rendelkezésünkre bocsátani;
- a lekérdezésszerűen összeválogatott adatsorokat képes csoportosítani (akár több szempont szerint is);
- az egyes csoportokban az aggregáló lekérdezésekhez hasonlóan képes összesítéseket (darabszám, összesítés, szélsőértékek) elvégezni.

Magát a jelentést kiegészíthetjük egyéb dolgokkal, például feliratokkal, képekkel is. A nyomtatásra mondunk néhány példát az online világból, hiszen az ottani adatbázisok is képesek jelentéseket készíteni. Ezek némelyikével már találkozhattunk korábban.

- Ma már otthonról vagy mobiltelefon segítségével bárhonnan lehet vonatjegyet vásárolni, és tíz éve még kinyomtattuk, de ma már egyszerűen csak letöltjük a telefonunkra, megspórolva ezzel a sorban állást a pénztárnál és meggátolva a vonat lekésését a hosszú sor miatt. Ez az úgynevezett e-vonatjegy vagy e-Ticket.
- Az áruházakban nyomtatott számla is jelentés, és ha az áruház rendszere praktikus, akkor a pénztárgép le is vonatja a raktárkészletből az általunk megvásárolt árukat, ráadásul a hamarosan elfogyó termékekről üzenetet küld a rendszer az üzletvezetőnek, hogy ne adódjon elő készlethiány.

- A Kréta is az adatbázisból készít jelentést, a félévi értesítő ennek a nyomtatott formája.
- Amikor repülővel utazunk valahová, a reptéren beszállókártyával kell jelentkeznünk, ez nagyjából olyasmi, mint a vonaton a helyjegy, ez igazolja, hogy megvettük a repülőjegyet az adott járatra.

Feladatok

1. Az *Alapfogalmak* című óra melyik zárófeladatához használnánk aggregáló lekérdezést?
2. A *Nem nyelvtanóra lesz, vagy mégis?* című óra konferenciás mintapéldái közül melyikhez használnánk aggregáló lekérdezést?
3. Gyűjtsük össze, hogy milyen kiszámított mezők lehetnek egy áruházi számlán!
4. Nézzük meg az interneten keresett képpel, hogy miként néz ki egy e-vonatjegy, és milyen információkat tartalmaz!
5. Az eddig felsoroltakon kívül milyen más jelentésekkel találkoztunk már? Ismertessük ezeket (pl. mozi-, színház- vagy koncertjegy, esetleg online számla valamelyik szolgáltatótól, például mobiltelefonostól)!

D A digitális eszközök használata

Mielőtt elkezdenénk…

„Ha egy fészekben van 10 kakukktojás, és egy fürjtojás, akkor melyik a kakukktojás?”

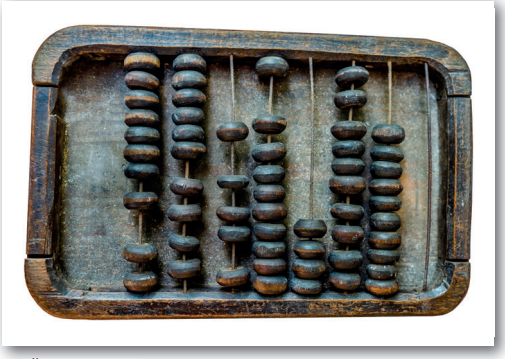
Ez a fejezet egy kakukktojás a könyvben. Az itt leírtak nem önálló tanulási egységei a tananyagnak, hanem ide akkor kell lapozni, ha más témakörhöz kapcsolódóan felmerül egy kérdés az eszközhasználattal kapcsolatban. Ezt a fejezetet először címek szintjén érdemes gyorsan áttekinteni, hogy tudjuk, mik azok a területek, amikről szükség esetén itt olvashatunk. Ha feladataink elvégzése közben felmerül egy kérdés, amiről itt van leírás, akkor az adott részt tanulmányozzuk, és használjuk fel.

Az egyes témák után ebben a fejezetben is található *Feladatok, kérdések* rész. Ha a téma felkeltette az érdeklődésünket, akkor az itt leírtaknak érdemes lehet máshol is utánanézni. Ezek nem kötelező részei a tananyagnak, ebben a könyvben a kérdésekre adandó válaszok nem olvashatók, de sok érdekességre lelhetünk, míg megtaláljuk azokat.

Egy kis történelem

Az informatikai eszközök története különböző szempontok szerint egészen eltérő időszakokat eredményez. Ha úgy nézünk ezekre az eszközökre, hogy mikroprocesszorokkal működnek, akkor történetük ott kezdődött, amikor megalkották az első mikroprocesszorokat. Így csak néhány évtizeddel ezelőttről beszélhetünk az informatikai eszközök történelme kapcsán. De ha azt tekintjük kiindulásnak, hogy az ember már nagyon régen törekedett olyan szerkezetek megalkotására, amelyek a távollétében az általa megadott folyamatokat elvégzik, akkor egészen az ősemberek csapdakészítéséig visszamehetünk az időben. Képzeljük el azokat az ősembereket, akik gödröt ástak, és ágakkal, gallyakkal fedték el, hogy az arra járó vadat elejthessék. Egy tökéletesen időzített automatát alkottak. Pontosan akkor működött, amikor a vad arra járt. Soha nem ejtette el az állatot korábban vagy később. Pont akkor, amikor arra járt. És ezt tette az ember távollétében. Ha innen közelítjük meg az automaták, az informatikai eszközök történetét, akkor százezer években mérhetjük azt.

Ha kevésbé szélsőségesen szeretnénk megközelíteni, akkor az informatikai eszközök történetét a számolás automatizálásának a történetével indíthatjuk. Az ókor különböző civilizációiban jelent meg az abakusz, mint a számolást könnyítő eszköz.

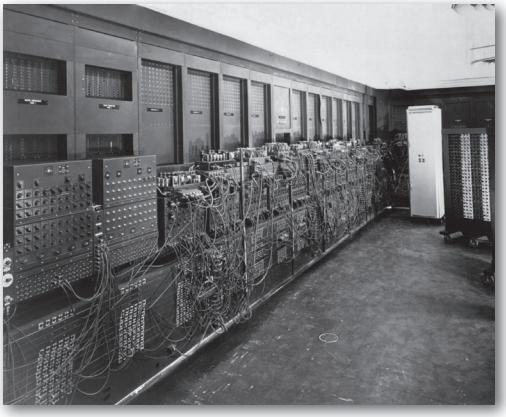


► Ősi abakusz

Mondhatjuk, hogy ezek voltak az első digitális számolóeszközök. Így néhány ezer évre nyúlik vissza az informatika története.

Megközelíthetjük onnan is, hogy a középkor végének és az újkor elejének számos neves tudósa mechanikus számológépet készített. Az 1600-as években Wilhelm Schickard, Blaise Pascal és Gottfried Wilhelm Leibniz is készített, tökéletesített ilyen eszközt. Így már csak néhány száz évre szűkítettük az informatika történetét.

A műszaki fejlődés a háborúk idején mindig lökésszerűen megugrik. Ez történt a II. világháborúban is. A különböző harci eszközök fejlesztése és használata nagy mennyiségű számítás elvégzését igényelte. Például egy új löveg elkészítése után szükség volt arra, hogy a lövegkezelő rendelkezzen egy olyan táblázattal, amiből kiolvashatja, hogy a löveg milyen állásában mekkora távolságra lehet vele lőni. Amikor rendre készültek az újabb és újabb lövegek, megnövekedett az igény a számítási kapacitások iránt, hogy a szükséges táblázatokat minél hamarabb és pontosabban elkészíthessék. Erre és hasonló problémákra építették az Egyesült Államokban az első tisztán elektronikus, általános célú digitális számítógépet, az ENIAC-ot az 1940-es években. Ezzel értünk el az informatika történetének szűkítésében oda, ahonnan már mondhatni egyenes ágon leszármaztathatóak a mai informatikai eszközeink, a laptopunk, a telefonunk, a fitnesz karperecünk, az intelligens mosógépünk, az okostévénk. Ezek az eszközök működésük, felépítésük alapján nagyon sok mindenben megegyeznek az ENIAC működésével, felépítésével. Persze tudjuk, hogy az ENIAC teremnyi méretű berendezés volt, ezzel szemben egy okosóra három–négy nagyságrenddel gyorsabban végzi a számítási műveleteket, de mégis az alapelvek szintjén rengeteg az azonosság.



► ENIAC az első általános célú elektronikus digitális számítógép

Feladatok, kérdések

1. Mi motiválhatta Blaise Pascalt a mechanikus számológépe megépítésében? Érdemes több független forrást is keresni, hogy lehetőség szerint hiteles információkhoz jussunk.
2. Gyűjtsünk adatokat az ENIAC számítógépről! Mekkora volt az alapterülete? Mekkora volt a tömege? Mekkora volt az áramfogyasztása? Hány összeadási műveletet tudott másodpercenként elvégezni? Mai értékre átszámítva mennyibe került az előállítása?
3. Hasonlítsuk össze egy mai számítógép processzorában található tranzisztorok számát az ENIAC gépben található elektroncsövek számával!

A modern digitális eszközök működése

Nézzük azokat az alapelveket, amikben egységesnek tekinthetjük a mai informatikai eszközöket. Ehhez vissza kell nyúlnunk, az 1940-es évek közepéig, amikor az Amerikai Egyesült Államokban egy tudóscsoport azon dolgozott, hogy létrehozzon egy teljesen elektronikus, programozható számítógépet, az ENIAC-ot. A készülék korszakalkotó volt, de a tudósok, akik dolgoztak rajta, menet közben már látták, hogy mit és hogyan kellene módosítani, hogy egy még hatékonyabb, és univerzális gépet hozhassanak létre. Az ENIAC tapasztalatai alapján a Pennsylvaniai Egyetemen a kutatócsoport megállapításait a magyar származású Neumann János – aki a következő elektronikus számítógép (EDVAC) megépítését végző projekt tanácsadója volt – vetette papírra. Ezért mi magyarok szeretjük ezeket az elveket Neumann-elveknek nevezni, de fontos tudnunk, hogy ez egy csapat matematikus és mérnök közös munkája volt.

A teljesség igénye nélkül vegyük sorra ezeket az elveket:

▶ Elektroncső

- **A számítógép legyen teljesen elektronikus működésű!** Az ENIAC és az azt követő években a többi számítógép is elsősorban elektroncsövekből épült fel. Az így elkészített berendezések számolási sebessége akár az ezerszerese is volt a korábbi mechanikus alkatrészeket is tartalmazó gépeknek. Így a sebesség szempontjából is egyértelmű volt, hogy ez a jövő. Ha megnézzük a mai digitális eszközök számítási sebességét, akkor azt látjuk, hogy ezt a sebességet sokmilliószorosan túlléptük azóta.
- **Az eszköz használja a kettes számrendszert a műveletvégzéseiben!** E mögött egy nagyon praktikus mérnöki gondolat húzódik meg. A kettes számrendszerben csak kétféle számjegy létezik (0 és 1), ami elektronikai eszközökkel könnyebben kezelhető állapotot jelent, mintha 10 értéket kellene minden helyiértéken megkülönböztetni, ahogy az a tízes számrendszerben szükséges. Mérnökileg egy alacsony és egy magas feszültségszint kezelése nagyobb toleranciát, hibatűrést, könnyebben megépíthető áramköröket jelent. A kettes és tízes számrendszer közötti átváltás egész számok esetében problémamentes, törtek esetében a véges darabszámú helyiértékek miatt adódnak átváltási, kerekítési problémák, amelyekre programozáskor figyelemmel kell lennünk. A kettes számrendszer használata az elektronikai összetevőkkel egyszerűbb áramköröket eredményez, mintha ugyanezt tízes számrendszerrel kellene megtenni, így ez az elv mind a mai napig meghatározó az informatikai eszközeinkben.
- **A gépnek legyen belső memóriája, ami az adatok és a programutasítások tárolását egyaránt elvégzi!** Ennek egyik következménye az, hogy a programutasításokat adat-

ként kezelve, azok módosíthatók, tehát a program képes akár önmagát is megváltoztatni. Az első számítógépek esetén néhány tíz, néhány száz adat tárolását oldották meg. Napjainkban több milliárd adatot tárolhatunk egy számítógép memóriájában.

- **A gép legyen univerzális, azaz ne egy speciális feladatra készüljön, hanem a programok segítségével különböző feladatokat legyen képes ellátni!** A mai informatikai eszközökben ezt a tulajdonságot teljesen természetesnek tekintjük például akkor, amikor a telefonunkra letöltünk egy új programot, ami olyan feladatokat lát el, amire korábban a telefonunk nem volt képes.
- **A számítógép legyen soros végrehajtású, azaz a program utasításai egymás után, időben sorban történjenek!** A soros végrehajtás teljesen praktikus okokból került az alapelvek közé. Ennek megvalósítása egyszerűbb volt, mint a párhuzamos programvezérlés, ez csökkentette az egyébként sem egyszerű berendezés bonyolultságát. Ez az elv az, ami a mai eszközöknél már sokszor nem teljesül. Egy mai korszerű számítógép többmagos processzorral és egy komoly videokártyával rendelkezik, így ezek az elvégzendő műveleteket egyszerre, párhuzamosan hajtják végre, és nem kapcsolják ki magukat addig, amíg a másik eszköz végzi a számításokat.

Feladatok, kérdések

1. Neumann János milyen iskolákat végzett, mely tudományterületeken ért el jelentős eredményeket?
2. Keressünk adatokat arról, hogy egy mai mobiltelefon és egy tíz évvel ezelőtti laptop számítási teljesítménye hogyan viszonyul egymáshoz!

A digitális eszközök főbb egységei

Sokféle digitális eszközzel vesszük magunkat körbe. Ezek jelentősen különbözőnek tűnnek, de ha megnézzük a belső felépítésüket, a funkcionális összetevőiket, akkor láthatjuk, hogy jobban hasonlítanak egymásra, mint elsőre gondolnánk.

Milyen részekből is áll egy asztali számítógép, egy laptop, egy okostévé, egy mobiltelefon, egy tablet, egy intelligens távirányító, egy e-book olvasó, egy okosóra, egy fitnesz karkötő?

▶ Laptop

Perifériák

Az elsők, amikkel a használat közben találkozunk, a **be- és kimeneti perifériák**. Ezek azok a részek, amiken keresztül mi információt tudunk adni az eszköznek, és amelyeken keresztül az eszköz tud válaszolni.

Szinte minden digitális eszköznek van egy **kijelző**je, **képernyő**je. Ez általában egy színes, grafikus megjelenítő. Itt egyik értékként a *képátló hosszát* szokták megadni, jellemzően inch-ben (inch: hüvelyk, 1 hüvelyk = 2,54 cm). Ez mobiltelefonnál lehet például 6” (~15,2 cm), egy asztali monitornál 24” (~61 cm), egy tévénél 55” (~140 cm). Másik jellemző érték a *felbontás*, azaz, hogy vízszintesen, függőlegesen hány képpontot tud megjeleníteni egymás mellett a kijelző. Egy monitor esetében lehet például 1 920 × 1 080 (FullHD), ami azt jelenti, hogy vízszintesen 1920 képpontot, függőlegesen 1080 képpontot tud megjeleníteni. Ez összesen 1 920 · 1 080 = 2 073 600 képpont a kijelzőn. Egy 4K-s tévé esetén ehhez képest mind vízszintesen, mind függőlegesen dupla annyi képpont van egy sorban, illetve oszlopban. Tehát 3 840 × 2 160 a felbontás, ami több mint 8 millió képpontot jelent a képernyőn. A legtöbb kijelző színes, a pixelgrafika fejezetben tárgyalt *RGB* színkódolást használ. Van néhány eszköz, aminek a kijelzője *fekete-fehér*, vagy csak a szürke néhány árnyalatát tudja megjeleníteni.

▶ Hagyományos könyvek és e-book olvasó

Ennek oka lehet, hogy sokkal olcsóbb egy egyszínű kijelző, és sok esetben elegendő is. Másik ok a technológia lehet. Az e-book olvasók az e-papír technológiája miatt csak *szürkeárnyalato*s megjelenítésre képesek.

Több olyan informatikai eszköz van, ami kijelző helyett, vagy mellett képes **hang** segítségével is információt adni nekünk. Itt a legegyszerűbb a *sípoló*, *csipogó*, *bippegő* hang, ami például a be- vagy kikapcsolást, egy érték elérését jelezheti, vagy egysze-

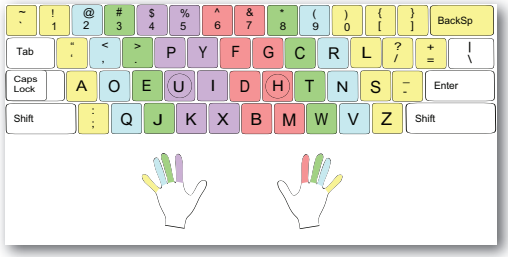
▶ Okostévé

rű figyelmeztetést adhat. Ennél összetettebb hangokkal is rendszeresen találkozunk, amikor zenét játszunk le, vagy például a digitális asszisztensünk tájékoztat az aktuális időjárásról, vagy futás közben az okostelefonunk egyik programja elmondja, hogy milyen sebességgel tettük meg az utolsó kilométert. A hangok esetén van, amikor elegendő, hogy hallunk egy információt, de zene esetén már szeretjük a *sztereó* hangzást, azaz azt, hogy a jobb és a bal oldalon akár eltérő lehet a hang. Ekkor már nem elegendő egy *hangszóró*, legalább kettőre van szükség. A számítógépes játékoknál, filmeknél a hangzás térbelisége, a mély hangok kiemelése (például egy akciójelenetben) még fontosabbá válhat. Ekkor már két hangszóró nem is elegendő. Ilyenkor a hangszórók körülvehetnek minket, és teljessé tehetik a *térbeli hangzás*t. Egy profi rendszerben akár 7–8 *hangcsatorna* is elkülönülhet.

Az utóbbi időben egyre több eszközben jelenik meg a *rezgőmotor*, amivel a zsebünkben lévő telefon, a csuklónkon lévő okosóra vagy fitnesz karkötő ad diszkrét jelzéseket.

Láthattuk, hogy az érzékszerveink közül a látás, a hallás, a bőrérzékelés használata már mindennapinak számít az informatikában a **kimeneti perifériák**nál, azaz azoknál az összetevőknél, amiknek a feladata az, hogy információt közöljenek velünk.

A kimeneti perifériák után vegyük sorra, hogy milyen módon tudunk mi információt adni egy digitális eszköznek. Egy számítógép esetén elsőként a *billentyűzet* juthat az eszünkbe. Ennek segítségével szövegeket tudunk beírni, utasításokat tudunk adni, vezérelni tudjuk a programok működését, irányítani tudjuk kedvenc karakterünket egy játékban. A billentyűzet használatakor fontos, hogy milyen nyelvhez készült a billentyűzet, mert ha magyar billentyűzethez szoktunk, akkor egy angol vagy francia billentyűzeten a speciális írásjelek megtalálása jelentős időbe kerülhet. Ha egy angol billentyűzet elé ültetnek minket, akkor a magyar ékezetes karakterekkel leszünk bajban. Találkozhattunk már azzal a helyzettel, hogy egy billentyűzeten egyes gombokat lenyomva nem az jelent meg a kijelzőn, mint ami a gombon volt. Ennek oka, hogy a számítógépes program határozza meg azt, melyik billentyű milyen karaktert is jelentsen. A *billentyűzet nyelve* az operációs rendszer beállításai között módosítható. Ezért amíg nem tudunk vakon gépelni, akkor dolgozhatunk jól egy billentyűzettel, ha a rajta lévő feliratok és az operációs rendszer billentyűzetre vonatkozó beállításai megegyeznek.



▶ Dvorak billentyűkiosztás

Ahogy a szövegszerkesztés fejezetben is olvashattuk, a billentyűzetkiosztás még a mechanikus írógépeken kialakított elrendezést követi. Az írógépeken úgy tették egymás mellé a billentyűket, hogy a szövegben gyakran egymás mellé kerülő betűk távol legyenek, mert az írást végző mechanikus karok így akadhattak legkevésbé össze. Egy számítógép

billentyűzete, vagy egy okostelefon képernyőjén megjelenő virtuális billentyűzet esetén ez az elrendezés így már nem indokolt. Sok kutatás foglalkozott azzal, hogy milyen elrendezés mellett lehetne sokkal gyorsabban gépelni. Az ezek alapján létrehozott billentyűzetek kísérleti jelleggel megjelentek, de elterjedni nem tudtak. Ennek elsődleges oka a megszokás, az attól való eltérés nehézsége.

Egy számítógép esetén a grafikus felhasználói felület kezelésének másik fontos eszköze az **egér**. Ha pontosabban és általánosabban akarunk fogalmazni, akkor a **mutatóvezérlő**, mivel ez lehet akár egér, érintőpárna, trackball (hanyattegér), pöcökegér.

Melyiket hol használjuk, mik az előnyei?

Egér: a legelterjedtebb mutatóvezérlő eszköz. Már biztosan találkoztunk több fajtával is, amiken eltérő számú gomb volt. A régebbieknek az alján golyó volt, az újabbakon optikai érzékelő figyeli a mozgást.

Érintőpárna (touchpad): a laptopokon láthatjuk a billentyűzet alatti területen. Ha a laptopot hordozható gépként használjuk, akkor ez a mutatóvezérlő mindig ott van velünk, nem igényel külön helyet a gép mellett.

▶ Érintőpárna – touchpad

Pöcökegér: ez egy kisméretű botkormány a billentyűzetbe elhelyezve. Aki sokat gépel, és néha van csak szüksége a grafikus kurzor mozgatására, az a keze felemelése nélkül irányíthat ezzel a kis eszközzel. Kell egy kis idő, hogy megszokjuk, de utána gépelés közben mindig kézre esik. Programozók közül sokan kedvelik.

▶ Pöcökegér – trackpoint

Trackball: szokták hanyattegérnek is nevezni, mert a régi egerek alján lévő, mozgás érzékeléséhez szükséges golyó itt fentre került. Ezt a golyót kell az ujjunkkal mozgatni, aminek hatására mozog a grafikus kurzor. Előnye az egérrel szemben, hogy ezt nem az asztalon kell mozgatni, és akkora helyre van csak szüksége, amekkora maga az eszköz. Sokan idegenkednek tőle, de a grafikusok között sokan használják.

▶ Hanyattegér – trackball

Ha már grafikusoknál tartunk, érdemes megemlíteni a *digitalizáló táblák*at is, amelyek egy kis táblán érintésérzékeny felülettel rendelkeznek, és egy speciális digitális íróeszköz tartozik hozzájuk. Ezen a felületen rajzolva a kép közvetlenül a számítógépben keletkezik, ahol a digitális toll a szoftver beállításaitól függően működhet például

▶ Táblagép (tablet)

ceruzaként, tollként, ecsetként, radírként. A nagyon precíz nyomásérzékelőnek köszönhetően a vonalrajzolás intenzitása gyengébb vagy erősebb ceruzarajzot eredményezhet, vagy az ecsetvonásokat tudja jól visszaadni.

Vannak olyan digitális eszközeink, amelyek grafikus felhasználói felülettel rendelkeznek, de se billentyűzetet, se egeret nem szoktunk hozzájuk kapcsolni. Ilyen például az okostelefon és a tablet (táblagép). Ezeknél a szöveg bevitelét és a grafikus kurzor pozícionálását a kijelzőbe épített érintésérzékeny felületen végezhetjük el. A gépeléshez ilyenkor egy virtuális billentyűzetet használunk.

Az eddig felsorolt bemeneti perifériákat mind a kezünkkel vezéreltük. Mozgással még más különböző információ beviteli lehetőségeink vannak. Gondoljunk arra, hogy amikor a mobiltelefont elforgatjuk 90 fokkal, akkor azt a képernyőn megjelenő program is képes követni. Például a böngészőben 90 fokkal elfordul a szöveg, vagy a lejátszott film is fordul. A telefonban, tabletben különféle mozgásérzékelők vannak, melyek képesek érzékelni az elmozdulás irányát, sőt akár a sebességét, gyorsulását is. Ugyanilyen érzékelőket használnak a fitnesz karkötőkben a mozgás érzékelésére, a lépés számlálására. Ezek a mozgási adatok adják meg feldolgozás után, hogy egy kirándulással, egy futással mennyi energiát égettünk el, mennyit nassolhatunk ezért cserébe anélkül, hogy tartanunk kellene az elhízástól.

A digitális eszközeink egyre nagyobb részét tudjuk már hanggal is vezérelni. Ehhez az adott nyelven „értő” szövegfelismerőre van szükség. Az egyszerűbb utasításoktól, mint például a mobiltelefonnál „Hívd Anyut!”, az összetett feladatokig, mint a szövegdiktálás, vagy a digitális asszisztensnek adott keresési feladatok, lehetőségünk van már akár magyarul is, hanggal vezérelni berendezéseinket. Nem meglepő módon ehhez egy mikrofonra van szükség, és természetesen arra a programra, ami képes a hangrezgésekből létrehozott elektromos impulzusokat, digitális jeleket szövegként, utasításként kezelni, feldolgozni.

A digitális eszközeink elsődlegesen a kijelzőn keresztül kommunikálnak velünk. Viszont nekünk is van lehetőségünk képeket bejuttatni az eszközbe. Ehhez egy kamerára van szükségünk. A *kamera* lehet a mobiltelefonba vagy laptopba beépített, de lehet egy számítógéphez külön csatlakoztatott *webkamera*, vagy akár egy *digitális fényképezőgép*, amit a számítógépünkhöz kapcsolunk. A kamera képénél fontos szempont, hogy milyen minőségű képet tud előállítani. Ezt sok összetevő mellett a kamera felbontása is erősen meghatározza. Egy laptopba integrált 1280×720 képpont felbontású kamera képe kevesebb képpontot biztosít, mint amennyit egy FullHD felbontású monitor képes megjeleníteni. Egy mobiltelefonba integrált 32 megapixeles (32 millió képpontos) kamera képe négyszer annyi képpontot tartalmaz, mint amennyit egyszerre meg tud jeleníteni egy 4K felbontású tévé. Miért van ekkora felbontásra szükség? Ha képernyőn szeretnénk a teljes képet meg-

jeleníteni, akkor nincs szükség ekkorára. Ha viszont szeretnénk egy részletet kinagyítani, kiemelni, akkor már fontos, hogy vannak olyan képpontok, amikkel ez megtehető. Ha nyomtatni szeretnénk, akkor nagyobb felbontású képre lesz szükségünk.

► Lapolvasó – scanner

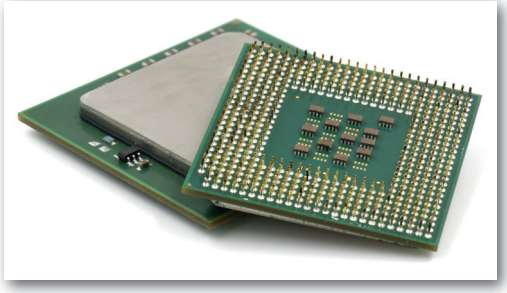
Ha dokumentumok, papírlapok tartalmát kell digitalizálnunk, akkor nem mindig tökéletes megoldás a digitális fényképezőgép vagy a mobiltelefon használata. Erre a feladatra sokszor megfelelőbb egy *lapolvasó* (scanner) használata. A lapolvasó felbontását nem úgy adják meg, hogy összesen hány képpontot képes rögzíteni, hanem úgy, hogy egy adott távolságon belül hány képpontot tud megkülönböztetni. Jellemzően a felbontást DPI (dot/inch: pont hüvelykenként) értékben adják meg, ami megmutatja, hogy 2,54 cm-en hány pontot tud megkülönböztetni a lapolvasó. Jellemző érték a 600 DPI, de egy otthonra is megfizethető árú berendezés akár 4800 DPI értéket is tudhat.

A képek digitalizálásának több módja és eszköze van. Gondoljunk csak a vonalkódolvasókra vagy a digitális röntgengépekre. Itt mindig a beolvasott képi információ feldolgozása lesz a fontos feladat, mint például egy bolti pénztárban a vonalkódolvasóval a vonalak által jelzett számsor értelmezése, a leolvasott termék adatbázisban tárolt árának hozzáadása a számlához, esetleg a raktárkészlet azonnali módosítása, szükség esetén az új termékek automatizált beszerzésének indítása.

Központi feldolgozóegység

Az eddigiekben azokról az eszközökről volt szó, amikkel a használat közben elsőként találkozunk, azaz a ki- és bemeneti perifériákról. Ezek csak a kapcsolatot teremtik meg a számítógép és a külvilág között. Nézzük, hogy mi van a „motorháztető” alatt!

A számítógép utasításokat hajt végre. Ezeknek az utasításoknak olyanoknak kell lenniük, amiket a gép belső alkatrészei megértenek, végre tudnak hajtani. Mi is hajtja végre ezeket az utasításokat? A digitális eszközünkön belül található egy, az utasítások végrehajtásáért felelős rész, ami egyrészt képes elvégezni a számítási műveleteket, másrészt képes a memóriában adatok formájában tárolt utasítássornak megfelelően vezérelni a számítógép működését. Ez a központi feldolgozóegység, angolul *central processing unit (CPU)*. Ez egy mikroprocesszor, amiben tranzisztorok segítségével létrehozott áramkörök felelősek a vezérlésért, a számítási műveletek elvégzéséért. A központi feldolgozó egységet röviden processzornak is szoktuk nevezni. A *processzor* napjainkban akár több, egymás mellett párhuzamosan működő egységből is állhat. Ebben az esetben ezeket a részeket processzor-magoknak nevezzük. Egy otthoni használatra szánt processzornak lehet 4–8

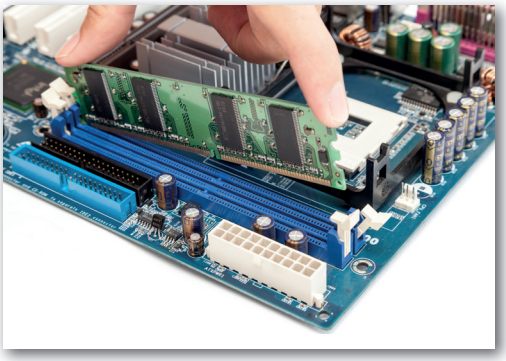


► Központi feldolgozóegység – CPU

magja, de speciális grafikai feladatok esetén ez akár 16 mag is lehet egy asztali számítógépben. Mivel ezek a magok külön-külön képesek egyidőben műveleteket végezni, ezért a Neumann-elvek között említett soros, azaz szigorúan egymás utáni feladatvégrehajtás már nem teljesül, illetve meghaladott.

Memória, RAM

A számítógép memóriája az, amiről mindenki hallott már, mindenki sejti, mi az, de sok esetben egy digitális eszköz leírásában zavaróan keverednek fogalmak, kifejezések, ami nem segíti a megértést. Szedjük sorra, hogy mi is az a memória, milyen lényeges fajtái vannak.



▶ Memória (RAM)

A számítógépnek szüksége van egy olyan adattároló részre, ami tárolja az éppen futó program utasításait, adatait, vagy azoknak legalább egy éppen feldolgozás alatt álló részét. Ennek egy olyan tárolónak kell lennie, amiből az adatok tetszőleges sorrendben, gyorsan elérhetők. Ez a napjaink digitális eszközeiben a **RAM** (Random Access Memory: véletlen elérésű memória). Ebben több milliárd bájtnyi adatot tudunk eltárolni. Egy asztali számítógépnél ez lehet 8, 16, vagy akár 32 GB is. Mobiltelefonoknál is találkozunk 10–12 GB-os értékekkel. Minél nagyobb ez az érték, annál több programot tudunk egyidőben futtatni, vagy annál nagyobb memóriaigénnyel rendelkező program képes akadásmentesen dolgozni. Ilyen lehet például egy videószerkesztési feladat. Az adatok a RAM-ban addig maradnak meg, amíg a RAM folyamatos áramellátást kap. Ha egy táblagép vagy mobiltelefon kijelzőjét kikapcsoljuk, akkor az az eszköz még működik, az akkumulátor kapacitásától függően akár napokig képes az adatokat megtartani. Ha lemerül az akkumulátor, akkor a nem mentett adatok elvesznek.

Háttértárak

Hova menthetjük az adatokat, ahol megmarad akkor is, ha már nincs áramellátás? A még mindig széles körben elterjedten használt **háttértár**-típus a merevlemezes meghajtó (HDD: Hard Disk Drive), ami egy zárt dobozban lévő, mágnesezhető lemezekre rögzíti az adatokat. Ez nem számít ma már elég gyorsnak, és a technológiából adódóan nem is nagyon lehet arra számítani, hogy érdemlegeset fejlődjön ezen a területen. Előnye az, hogy viszonylag alacsony áron biztosítja nagy mennyiségű adat tárolását. Jellemzően otthoni számítógépbe 2–4 TB kapacitású elegendő, de létezik 10–16 TB-os tárterületű is, ami például nagy mennyiségű videó tárolására alkalmas.

▶ Merevlemezes meghajtó (HDD)

Az egységnyi adat tárolásának költsége a merevlemezhez képest drágább az **SSD**-k (Solid-state drive: szilárdtest-meghajtó) esetén. Ezek a félvezető alapú tárolók nem tartalmaznak mozgó alkatrészeket, az adatok írása és olvasása akár több mint hússzorosa is lehet a HDD-k sebességének. Az ilyen típusú tárolók hasonlóan működnek, mint a korábban bemutatott memóriák, ezért több helyen ezeket is egyszerűen RAM-ként jelzik. Felmerülhet a kérdés, hogy amennyiben a memóriához hasonlóan működnek, de nem veszítik el tartalmukat az áramellátás kimaradása esetén, akkor miért nem ezeket használjuk a digitális eszközeink belső memóriájának. A válasz viszonylag egyszerű: több nagyságrenddel lassabbak, így program futtatására a mai sebességelvárások mellett nem lennének alkalmasak. A mobileszközökben – mint táblagép és mobiltelefon – ilyen típusú háttértárakat alkalmaznak. Ott ezek mérete elérheti a GB-os kategóriát, bár az eszköz árára ennek egészen jelentős hatása van. A mobil eszközök leírásánál sokszor a belső memóriát és a háttértárat is RAM-ként jelölik, ami nem segíti az eligazodást. Ha látunk két számértéket a RAM jelzés után, akkor szinte biztos, hogy ezek közül a kisebb érték a belső memória, míg a nagyobb számérték a háttértár.

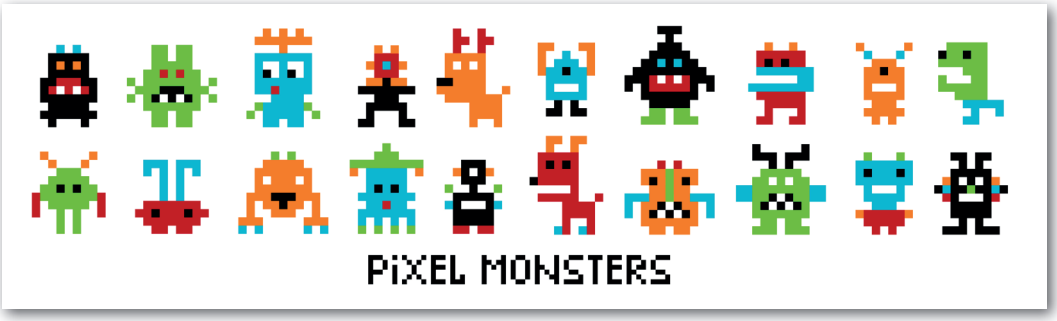
► Félvezető alapú tároló (SSD)

Egyéb kiegészítők

A fentebb leírt összetevőkből összeállítható nagyon sokféle digitális eszköz, de van pár olyan funkció, amihez külön speciális áramköröket, mikroprocesszorokat terveznek. Ennek oka, hogy az adott szolgáltatás ne terhelje a CPU-t, vagy adott területen speciális szolgáltatást tudjanak biztosítani.

Ma már minden eszköznél elvárt, hogy az internetre legyen kapcsolva. Az okosotthon koncepció szerint a lakásban lévő légkondicionáló, fűtésvezérlésért felelős termosztát, robotporszívó, hűtőgép és egyéb berendezések legyenek elérhetők interneten keresztül, azokat a mobiltelefonunkról vezérelhessük, legyünk bárhol a világon. Ehhez arra van szükség, hogy mindegyik eszközünk képes legyen hálózati kapcsolatot létesíteni vezetékes vagy vezeték nélküli módon. Ezt egy speciális, erre kifejlesztett áramköri elemmel oldják meg, amit sok esetben *hálózati kártyá*nak hívnak, holott nem feltétlenül van kártya alakja. Az elnevezés használata az 1980-as évek elejére nyúlik vissza, amikor az IBM PC (Personal Computer: személyi számítógép) létrehozásakor az volt az alapelv, hogy ezek a számítógépek az alapfelépítésük mellett legyenek bővíthetőek speciális feladatokat ellátó komponensekkel. Ezeket hívták bővítőkártyáknak. Amennyiben a hálózathoz vezeték nélküli kapcsolattal képes csatlakozni egy eszköz, szükséges egy antenna is, amit sokszor az eszköz dobozán belül helyeznek el, mivel a mérete ezt lehetővé teszi. Egy mobiltelefon képes többféle vezeték nélküli kapcsolatot létrehozni. Gondoljunk arra, hogy az utcán történő telefonáláshoz a telefontársaság átjátszóállomásához kell kapcsolódni, otthon wifikapcsolaton keresztül érjük el az internetet, a vezetéknélküli hangszóróhoz, fitnesz karkötőhöz pedig Bluetooth segítségével tudunk kapcsolódni.

Amikor egy filmet nézünk, egy számítógépes játékkal játszunk, a képi hatás mellett fontos számunkra, hogy a hangzás is teljes legyen, ne csak a kijelző irányából jöjjön, hanem



► 8 bites játék szereplői – alacsony grafikus teljesítmény mellett is lehetnek hőseink, antihőseink. :-)

vegyen minket körbe. Ehhez speciális többcsatornás hangrendszerre van szükségünk, aminek vezérlését egy hangkártya fogja elvégezni. Ez határozza meg, hogy hány hangszórót tudunk csatlakoztatni, milyen speciális hangeffektusokkal tudjuk kiegészíteni a hangzást.

A képi megjelenítésnél elvárjuk a nagy felbontást, az élethű színeket, az akadásmentes mozgást, a 3D élményt. A nagy számítási teljesítményre felkészített videokártya fogja biztosítani, hogy az operációs rendszer ablakkezelése látványos legyen, és egy videójátéknál a szereplők játékhelyzethez igazodva, de lehetőleg élethűen jelenjenek meg. Egy komoly, játékosoknak szánt videokártya ára egy számítógépben lehet akkora összeg, amennyiért egy szerényebb, de a legtöbb feladatra bőségesen elegendő grafikus képességgel megáldott komplett számítógépet lehet kapni.

Egy hordozható eszközben – laptop, táblagép, mobiltelefon – az utólagos bővítés lehetősége nagyon korlátozott. Egy laptopban cserélhető esetleg a memória vagy a háttértár nagyobbra, de például a processzor, vagy a beépített vezetékes hálózati csatlakozás nem cserélhető nagyobb teljesítményűre. Egy táblagépben vagy mobiltelefonban ennyi fejlesztési lehetőség sincs. A legtöbbször a háttértár bővíthető egy-egy SD-memóriakártyával.

Feladatok, kérdések

1. Mit jelent, ha egy kerek kijelzős okosórához azt írják, hogy a felbontása 480 × 480?
2. Hogyan működik az e-papír, miben különbözik egy OLED kijelzőtől?
3. Egy e-book olvasóba hány könyv fér el?
4. Mit jelent, hogy egy hangrendszer 2.0, 2.1, 5.1, vagy 7.1 jelzést kap?
5. Mi a giroszkóp feladata?
6. Nézzünk utána, hogy egy laptop CPU-ja mekkora alapterületű, és hány tranzisztort tartalmaz!
7. Mit jelentenek egy okostelefonnál a következő jelölések: RAM 8 GB / 512 GB, 6,2”, 64 MP / 12 MP, 2,8 GHz Octa-core CPU, IP68?

Operációs rendszerek

Az elektronikus számítógépek első generációját azok a tudósok használták elsősorban, akik a gép megépítésében részt vettek, vagy legalábbis értették a gép működését, felépítését. Az általuk készített programok teljesen az adott számítógép fizikai összetevőire, hardverére épültek. Később egyre többen kezdtek használni számítógépeket, és egyre nagyobb igény merült fel egy olyan felhasználói felületre, amin keresztül az ember könnyebben tud a számítógéppel kommunikálni. Ez kezdetben kizárólag karakteres felületen történt (angolul command line interface, röviden CLI). A személyi számítógépeknél az 1980-as években megjelentek a grafikus felhasználói felületek (angolul graphical user interface, röviden GUI), amelyek már széles kör számára tették könnyebben elérhetővé a számítógépek szolgáltatásait.

```
raerek@laptop:/$ ls -l

drwxr-xr-x  1 root root   4096 Oct 28  2018 bin
drwxr-xr-x  1 root root   4096 Jul 25  2018 boot
drwxr-xr-x  1 root root   4096 Apr 26 10:39 dev
drwxr-xr-x  1 root root   4096 Apr 26 10:39 etc
drwxr-xr-x  1 root root   4096 Oct 29  2018 home
-rwxr-xr-x  1 root root 591344 Jan  1  1970 init
drwxr-xr-x  1 root root   4096 Jul 25  2018 lib
drwxr-xr-x  1 root root   4096 Jul 25  2018 lib64
drwxr-xr-x  1 root root   4096 Jul 25  2018 media
drwxr-xr-x  1 root root   4096 Apr 20 11:40 mnt
drwxr-xr-x  1 root root   4096 Jul 25  2018 opt
dr-xr-xr-x  9 root root      0 Apr 26 10:39 proc
drwx------  1 root root   4096 Jul 25  2018 root
drwxr-xr-x  1 root root   4096 Apr 26 10:39 run
drwxr-xr-x  1 root root   4096 Feb 24 16:07 sbin
drwxr-xr-x  1 root root   4096 Jul 19  2018 snap
drwxr-xr-x  1 root root   4096 Jul 25  2018 srv
dr-xr-xr-x 12 root root      0 Apr 26 10:39 sys
drwxrwxrwt  1 root root   4096 Apr 20 11:41 tmp
drwxr-xr-x  1 root root   4096 Jul 25  2018 usr
```

▶ Linux ls parancs és eredménye – karakteres felhasználói felület

A **karakteres felület**en jellemzően utasításokat, parancsokat adhatunk ki, amiknek eredményeként szöveges válaszokat kapunk a képernyőn. Ehhez ismernünk kell a kiadható utasításokat, azok használatának módját, paramétereit, és néha az eredmény értelmezése is csak megfelelő előismeretek birtokában lehetséges. Ezzel szemben a mai **grafikus felhasználói felület**ek kialakításánál törekednek arra, hogy könnyen érthető, kevés előismerettel is használható legyen, a még esetleg nem ismert funkciók is intuitív módon felfedezhetők legyenek. A felhasználók széles körének készített operációs rendszerek szinte kivétel nélkül grafikus felhasználói felülettel rendelkeznek. Számítógépen ilyen például a *Windows*, a *macOS*, a *Chrome OS* és a különböző *Linux*ok. A mobiltelefonokon, tableteken például az *Android*, az *iOS*, valamint az *iPadOS*.

A különböző grafikus felhasználói felületek használata attól lesz a felhasználóknak könnyű, hogy nagyon sok elemükben azonos módon működnek, így az egyik megismerése segíti a másik használatát. Nézzünk pár példát! A felhasználói felületen az alkalmazások *ablak*ban futnak, azaz a kijelző jól meghatározott területét használják. Persze ez adott esetben lehet akár az egész képernyő is. Az ablakokban általában *menü*ket találhatunk, a menüpontokhoz almenüpontok tartozhatnak. Ha valamit be kell írnunk, akkor *beviteli mező*t kell kitöltenünk, döntéseinket *gomb*okkal jelezhetjük. Amennyiben egy rendszerben egyidőben több programot futtathatunk, akkor lehetőségünk van közöttük váltani, vagy sok esetben akár több program képét is láthatjuk egymás mellett. Az említett és más hasonló tulajdonságok miatt mondhatjuk, hogy könnyű ezeknek a rendszereknek a használatát elsajátítani. Ha valaki egy teljesen új rendszert szeretne készíteni, akkor figyelembe kell vennie, hogy a felhasználóknak ezek a már megszokott felületek, jelentősen eltérni ettől nagy bátorság és kockázat.

A grafikus felhasználói felület kezelésének egyik szinte elengedhetetlen eszköze az *egér* vagy más *mutatóvezérlő eszköz*. Az operációs rendszerek viszonylag egységesek az egér-

használatban. Vegyük sorra, mik az alapvető egérműveletek azon túl, hogy az egér segítségével a kis nyilat lehet mozgatni a képernyőn.

Bal gombbal *egy szimpla kattintás*: ez a kiválasztás, a kijelölés, ami lehet például egy gomb, egy felirat, egy állomány. Az érintőpárnán vagy érintőképernyőn ezt egy koppintással érjük el.

Bal gombbal *dupla kattintás*: ez a kiválasztott állomány megnyitása, a kiválasztott program elindítása, mappáknál a mappába belépés. Az érintőpárnán vagy érintőképernyőn dupla koppintással érjük el.

Jobb gombbal kattintás: ez jellemzően a helyi menüt hozza elő, ami az adott környezetben leginkább releváns műveletekhez ad gyors hozzáférést. Ezt érintőpárnán a legtöbb eszköznél kétujjas koppintással, míg érintőképernyőn ujjunkat hosszabban ott tartva érhetjük el.

Vonszolás, azaz amikor az egér bal gombját lenyomjuk, és miközben nyomva tartjuk, mozgatjuk az egeret: ez egy objektum, ablak áthelyezését teszi lehetővé. Az érintőpárnán vagy érintőképernyőn ez dupla koppintással érhető el úgy, hogy a második koppintás után nem emeljük el az ujjunkat, hanem mozgatjuk a felületen. A vonszolást a gomb vagy érintőfelület elengedésével fejezhetjük be.

Egyes programokban a többszörös kattintásnak is van funkciója. Próbáljuk ki például a szövegszerkesztőben, mi történik, ha egy szövegen belül egy szóra egyszer, kétszer, háromszor vagy négyszer kattintunk!

Természetesen a balkezes egérhasználathoz a jobb és a bal gomb felcserélhető az operációs rendszer beállításaiban.

Az egér mellett a billentyűzetnek is jelentős szerepe van az operációs rendszer kezelésében. Azért, hogy könnyen megtanulható legyen a különböző programok használata, a billentyűkombinációk sokszor ugyanazt, vagy nagyon hasonló feladatokat látnak el. Nézzünk néhány gyakran használt gyorsbillentyűt!

Ctrl + C: másolás. Ez sok esetben a Ctrl + Insert segítségével is elvégezhető.

Ctrl + V: beillesztés. Sokszor lehet helyette Shift + Insert.

Ctrl + X: kivágás. Esetleg Shift + Delete.

Ctrl + A: mindent kijelöl (A: all, magyarul összes).

F1: a súgót nyitja meg.

F2: átnevezés. A fájlkezelőben a fájl vagy mappa átnevezésére szolgál, a táblázatkezelőben pedig a kijelölt cella szerkesztésére.

Alt + F4: ablak bezárása

Ctrl + F4: egy ablak egy fülének bezárása, például böngészőben.

F5: frissítés. Amikor egy ablakban megjelenített tartalomról tudjuk, hogy az már megváltozott, de még a gép nem jelenítette meg, akkor ezzel frissíthetjük a tartalmat. Például a fájlkezelőben egy hálózati mappa megtekintésekor lehet jó, vagy a böngészőben, ha szeretnénk a weboldalt ismét betölteni, frissíteni.

F10: a program menüjének elérése.

Ctrl + S: mentés (S: save, magyarul mentés).

Ctrl + P: nyomtatás (P: print, magyarul nyomtatás).

Azzal, hogy a különböző programok hasonlóan viselkednek, ezáltal „kézre esnek", a szolgáltatásiak megvalósítják a könnyű, kényelmes használhatóságot, azaz a **szoftver er-**

gonómiát. Ehhez még hozzátartozik a felület logikus elrendezése, átláthatósága, a program által megvalósítandó funkciók logikus elérése.

Az operációs rendszer feladata a felhasználóval való kapcsolattartás, és a programok futtatásához szükséges környezet biztosítása, a programok futtatása. A leggyakrabban használt operációs rendszerek lehetővé teszik a programok párhuzamos futtatását, a váltogatást közöttük. Ezt ma már teljesen természetesnek tekintjük.

Egy számítógépnél megszokott, hogy a gép indulásakor ki kell választani, hogy melyik felhasználó fogja használni, azaz minden felhasználónak elkülönített *felhasználói fiók*ot hozhatunk létre. Ezzel elérhető, hogy legyen olyan tárterület a gépen, amit csak az egyik vagy csak a másik felhasználó érhet el. Ez az adatvédelem szempontjából nagyon fontos. Hasonlóan a tárterület biztosításához, felhasználónként azt is szabályozhatjuk, hogy kinek van lehetősége például programokat telepíteni, kinek csak futtatni. Akár az is megadható, hogy melyik felhasználó melyik programot indíthatja el.

A mobiltelefonokat jellemzően mindig csak egy személy használja, így ott a több felhasználó felvétele a rendszerbe nem alapvető elvárás. Vannak olyan mobiltelefonok, ahol van lehetőség több felhasználói fiókot is létrehozni, és vannak olyanok is, amelyeken egy felhasználónak az üzleti és magánjellegű tevékenységeihez kapcsolódó adatokat – mint például a fényképeket, e-maileket – lehet jelszóval védetten elkülöníteni.

Alkalmazások telepítése

Rendszeresen találkozunk a következő kifejezésekkel: alkalmazás, applikáció vagy röviden app, program, szoftver. Ezek sok esetben egymás szinonimáiként jelennek meg. Bár van közöttük néha különbség, most tekinthetjük ezeket azonos jelentésűeknek. Ezek telepítéséről lesz szó a következőkben.

Az operációs rendszerek biztosítják számunkra a programok futtatását, a felhasználóval a kapcsolattartást, a fizikai eszköz erőforrásaihoz való hozzáférést. Ezen funkciókon kívül napjainkban egy halom olyan programot is kapunk az operációs rendszerrel együtt, ami szigorúan véve nem része annak, de a rendszer használatbavételét nagymértékben egyszerűsíti. Ilyen lehet például egy szövegszerkesztő a laptopon, egy fényképezőalkalmazás az okostelefonon, egy lépésszámláló az okosórán.

Amikor egy digitális eszközt, például laptopot, mobiltelefont, tabletet, okosórát használunk, általában szükségünk van az alapfunkciókon kívül valami speciális feladatot ellátó szoftverre. Ilyen lehet például egy táblázatkezelő, egy levelezőprogram, egy azonnali üzenetküldő alkalmazás. Ezek a programok nem települnek mind a digitális eszközünkre az operációs rendszerrel együtt. Egyrészt azért, mert ezek mindegyikére nincs mindenkinek szüksége, másrészt közülük több nem ingyenes szoftver, azaz ki kellene fizetni az árát akkor is, ha nem is szeretnénk használni. Akkor hogyan juthatunk olyan szoftverhez, amire szükségünk van? Mivel már a programozás alapjait megismertük, erre jó válasz lehet, hogy írunk magunknak egyet. Mivel egy program kifejlesztése sok munkába, ezáltal sok időbe telik, egy-két egyszerű esettől eltekintve ez nem lesz jó általános megoldásnak. Szükségünk lesz szoftverfejlesztők által készített programokra.

A szoftverek ára

A szoftverek kifejlesztése időt igényel. Amennyiben a szoftver írója szeretné, hogy a programírásra fordított idejét kifizessék, akkor az elkészített termékért pénzt kell kérnie. Ennek több módja is van, így előfordul, hogy egy számunkra ingyenes szoftverért a fejlesztője mégis kap pénzt. Nézzük a legáltalánosabb módjait ennek!

A programnak van egy ára, amit ki kell fizetnünk a termék beszerzésekor. Ekkor rendszerint az ár egy részét a termék forgalmazója, mint kereskedői jutalékot teheti el, de a szoftver fejlesztője is részesül a bevételből.

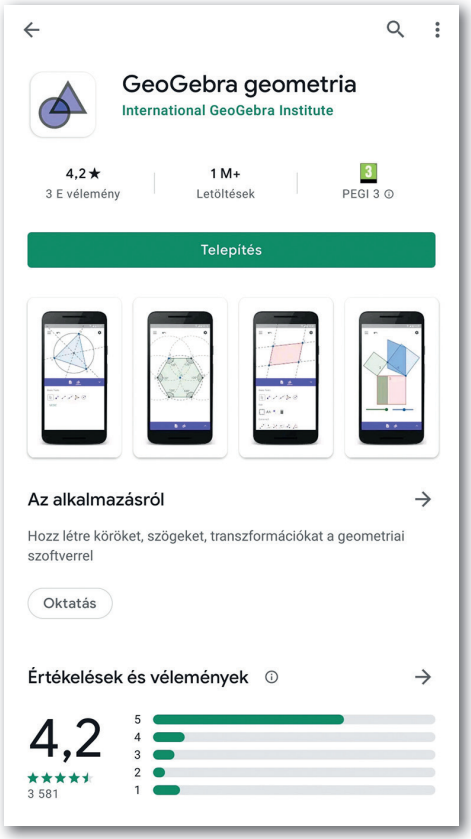
A programok egy másik köre számunkra ingyenes. Mitől is lehet ez ingyenes? Előfordul, hogy az otthoni vagy oktatási felhasználást ingyenesen biztosítják, míg az üzleti felhasználásért kérnek pénzt. Ekkor az üzleti célú felhasználás bevételéből végzik a fejlesztést, és mondhatjuk, „jófejségből” biztosítják a többi felhasználás ingyenességét. Ez azért egyfajta reklám a terméknek, mert aki magáncélra szívesen használ egy programot, nagy eséllyel megvásárolja, ha az üzleti tevékenységéhez is erre volna szüksége.

Lehet, hogy egy program csak egy ideig ingyenes, vagy a funkcionalitása korlátozott, amíg nem fizetünk érte. Ebben az esetben kipróbálhatjuk az alkalmazást, és amennyiben tetszik nekünk, úgy gondoljuk, hogy érdemes beszereznünk a korlátozásoktól mentes változatot, fizetnünk kell érte.

A mobileszközökön használható szoftvereknél elterjedt a termékbe integrált reklámozás. Így a szoftverfejlesztő azért kap pénzt, hogy megjelenítse a reklámokat minél több embernek. Ekkor a szoftvert használó ingyen használhatja az appot, de el kell viselnie a reklámok megjelenéséből adódó kellemetlenséget. Erre az esetre is szokták biztosítani, hogy ha fizetünk a szoftverért, akkor annak a reklámmentes változatát használhatjuk.

Az is előfordulhat, hogy az állam, minisztérium, iskola vásárolja meg a jogot, hogy a hozzá tartozó diákok használhassanak egy szoftvert, ami a tanulmányaikat segíti. Ebben az esetben a diáknak a szoftver használata ingyenes, de a terméket valaki mégis kifizeti. Ilyenkor a mennyiségi kedvezmény miatt jelentősen, akár nagyságrendekkel is olcsóbbá válhat a szoftver használatának joga, mint ha mindenki egyesével vásárolná meg magának.

Előfordul, hogy egy alkalmazás egy rendszer része, ahol a rendszer használatát fizeti ki valaki, de a hozzá tartozó alkalmazást a felhasználók ingyen telepíthetik és használhatják. Erre jó példa az elektronikus napló, ahol a diákok, szülők, tanárok ingyenesen telepíthetik eszközeikre azt az alkalmazást, amin keresztül hozzáférnek a napló adataihoz.



▶ GeoGebra telepítése Android telefonra a Google Play áruházból

A szoftver árának kifizetése

Felmerül a kérdés, hogy ha egy szoftverért fizetni kell, hogyan tehetjük azt meg. Bár kevésbé elterjedten, mint régebben, de még ma is létezik a boltban, személyes jelenléttel történő szoftvervásárlás. Erre az egyik jellemző példa, amikor az operációs rendszert vesszük a számítógéppel együtt, és a gép ára mellett megjelenik egy külön tételként a szoftver ára is a számlán. Másik jellemző példa, amikor egy számítógépes játékhoz a telepítő DVD-t vesszük meg számítógépünkhöz vagy játékkonzolunkhoz. Ezekben az esetekben a teljesen klasszikus megoldások, mint a készpénz vagy bankkártyás fizetés jelenik meg. Előfordulhat az is, hogy a vételárat banki átutalással fizetjük ki, de ez inkább az üzleti ügyfeleknél jellemző.

A szoftvereket egyre elterjedtebben online vásároljuk meg. Ez azt jelenti, hogy a fizetést is online intézzük, és a megvásárolt programot sem kapjuk meg telepítőlemezen, hanem csak a letöltéséhez szükséges linket. Ilyenkor szinte kivétel nélkül valamilyen bankkártyás fizetési lehetőséget biztosítanak. Akinek nincs bankkártyája, az így nem tud vásárolni, ilyenkor lehet megkérni például a szülőket, hogy a terméket vásárolják meg számunkra saját bankkártyájukkal. Fontos, hogy a bankkártyaadatainkat csak megbízható weboldalon

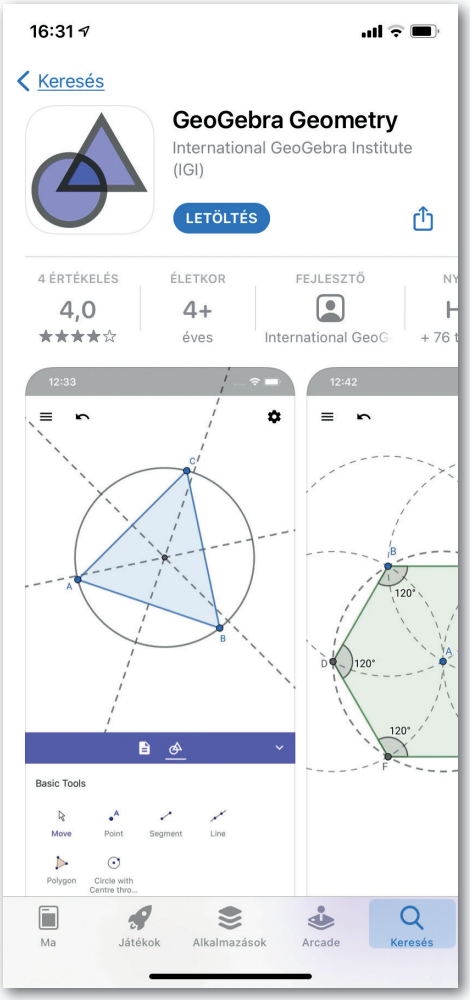
adjuk meg, azaz a legjobb, ha a webáruház a fizetéshez átirányít minket egy ismert bank titkosított, hitelesített fizetési oldalára.

Napjainkban elterjedtek az operációs rendszerekhez kötődő alkalmazásáruházak, ahol beszerezhetjük az ingyenes vagy fizetős szoftvereket. Ilyen például a Windows esetén a Microsoft Store, az Android esetén a Google Play, az iOS esetén az App Store, vagy az Ubuntu Linux estén az Ubuntu Software. Itt a szoftverekért – amennyiben szükséges – a letöltéskor a megadott bankkártyával tudunk fizetni. Illetve, amennyiben a rendszer támogatja a családi csoportokat, van arra is lehetőség, hogy a szülő által feltöltött egyenleg keretéig a gyermek vásároljon alkalmazást. A szoftverekért lehet, hogy egyszeri díjat kell fizetni, de amennyiben egy online szolgáltatás is kapcsolódik hozzá, akkor havi vagy éves díjat kell fizetni. Ez utóbbi esetben a szoftvert, a szolgáltatást addig használhatjuk, ameddig fizetünk érte.

Az alkalmazás beszerzése

A programok beszerzésének több módja is van. Az egyik a korábban már említett bolti vásárlás, amikor úgynevezett dobozos terméket veszünk. Amennyiben megbízható boltban, eredeti csomagolásban vásároljuk meg a terméket, nem kell aggódnunk a programmal kapcsolatosan. Ha láthatóan másolt változatot kapunk, akkor nagy eséllyel nem lesz jogtiszta a termék, azaz hiába is fizettünk érte, a szoftver gyártója hamisított, illegális változatnak fogja tekinteni a példányunkat. Hasonló eset áll elő, ha online módon nem a szoftver készítőjétől, vagy annak felhatalmazott kereskedőjétől vásároljuk meg azt.

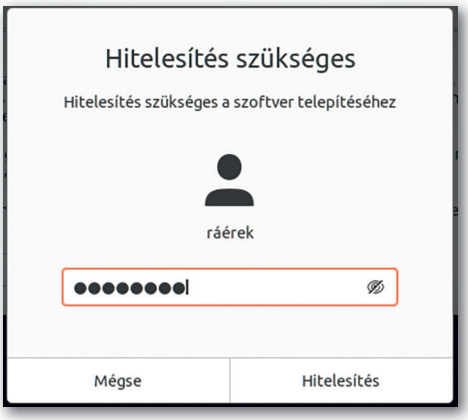
Az operációs rendszer alkalmazásáruházából beszerzett programok esetében a termékért a fizetés is jellemzően az áruház rendszerén keresztül történik. Ilyenkor a szoftverfejlesztők az adott alkalmazásáruházat mint kereskedőt bízzák meg az alkalmazás értékesítésével. Nagyon sok program esetében elkészítik annak különböző változatait az egyes operációs rendszerekre. Ilyenkor keresztplatformos fejlesztésről beszélünk.



► GeoGebra letöltése iPhone-ra az App Store áruházból

A programok telepítése

Ahhoz, hogy szoftvert telepíthessünk egy digitális eszközre, szükségünk van egy speciális jogra, ami ezt lehetővé teszi. Ez asztali operációs rendszereknél hagyományos telepítéskor – mint például Windows esetén .exe vagy .msi állományból indított telepítés – rendszergazdai jogokat jelent. Ha egy számítógépet otthon egyedül használunk, akkor jellemzően az adott gépen rendszergazdai jogaink vannak. Vállalati, iskolai környezetben az alkalmazottak, diákok az általuk használt gépre nem kapnak rendszergazdai jogokat, hogy a szoftverek telepítései a vállalat, iskola rendszergazdái által ellenőrzöttek maradjanak. Így a meglévő programok mellé újabbakat önállóan nem feltétlenül tudnak telepíteni. Az operációs rendszerekhez tartozó alkalmazásboltokból jellemzően minden felhasználó saját maga számára tud telepíteni szoftvereket.



▶ A programok telepítéséhez sok esetben rendszergazdai jogok szükségesek

Nézzünk egy példát a szoftverek telepítésére a különböző operációs rendszerek alatt! A matematika tanulásához jól használható, ingyenesen telepíthető GeoGebra alkalmazás több platformon is elérhető.

Mobiltelefonra a telefon operációs rendszeréhez tartozó alkalmazásboltba kell belépnünk. Lépjünk be az App Store vagy a Google Play alkalmazásba! Itt a keresőfelületen adjuk meg, hogy GeoGebra. A megjelenő listából válasszuk ki a számunkra szükséges alkalmazást. Ilyenkor fontos megnézni, hogy az adott alkalmazást melyik cég készítette, mert sokszor találkozhatunk megtévesztően hasonló nevekkel, amivel igyekeznek minket olyan program telepítésére rávenni, amit nem is szerettünk volna. Az apphoz mindig tartozik egy leírás, amiben a szoftver készítője leírja, hogy mit tud, és milyen szolgáltatásokat nyújt a programja. Sokszor itt már el tudjuk dönteni, hogy esetleg nem is erre a szoftverre lenne szükségünk, mert nem tudja az általunk kívánt szolgáltatást nyújtani.

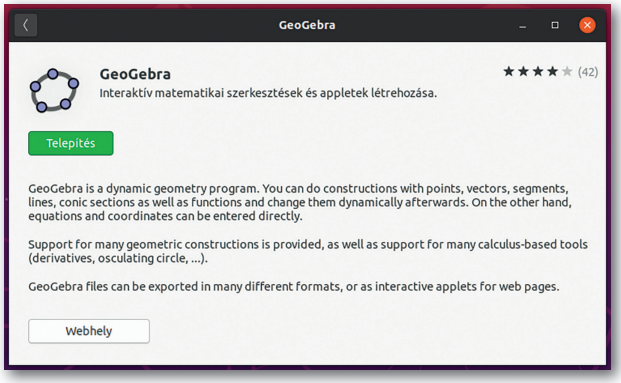
A programok mellett láthatjuk, hogy az adott alkalmazást hányan töltötték már le, valamint azt is, hogy milyen átlagos értékelést kapott egy ötfokozatú skálán. Érdemes ezt az értéket figyelembe venni, főleg, ha kimondottan nagyszámú értékelés eredményeként adódik. Az alkalmazásokhoz véleményt is írhatunk, ahogy ezt mások is megtették. Ezeket a véleményeket is érdemes telepítés előtt elolvasni, mert megtudhatjuk, hogy az általunk választott program egyes szolgáltatásairól mi a többiek véleménye, mennyire tartják jól használhatónak, esetleg mennyire zavaróak a program használata közben megjelenő reklámok. Sokszor azt is megtudhatjuk ezekből a bejegyzésekből, hogy probléma esetén a szoftver készítői mennyire segítőkészek, egy esetleges hibát mennyi idő alatt javítanak ki. Ezek mind fontosak lehetnek számunkra, mert a későbbi használati élményünket vetítik előre. Ha egy szoftvert több tízezer felhasználó együttesen például hármasnál gyengébbre értékelt, akkor ne legyenek illúzióink arról, hogy majd nekünk biztosan jó és hasznos társunk lesz.

Ha szeretnénk mi is véleményt írni egy programhoz, akkor azt mindig úgy fogalmazzuk meg, ahogy mi is hasznosnak látnánk azt másoktól. Ahol lehet, legyünk objektívek, a véle-

ményünket indokoljuk. Például nemtetszésünk esetén ne csak annyit írjunk, hogy a program rossz, hanem röviden fogalmazzuk meg, miben nem kaptuk azt, amit vártunk.

Ha meghoztuk a döntést, és telepíteni szeretnénk az alkalmazást, akkor válasszuk a Telepítés vagy Letöltés lehetőséget. Azokért a szoftverekért, amelyekért fizetnünk kell, a következő lépésben tehetjük ezt meg. Ilyenkor jellemzően bankkártya adatokat kell megadni ennek érdekében. Ha már vásároltunk korábban, és elmentettük a kártyaadatainkat, akkor egy egyszerűsített folyamattal megoldható ez, de a fizetés jóváhagyását minden esetben meg kell tennünk. Nem fordulhat elő, hogy egy alkalmazás telepítésekor úgy vonnak le a kártyánkról díjat, hogy azt előre nem jelzik nekünk.

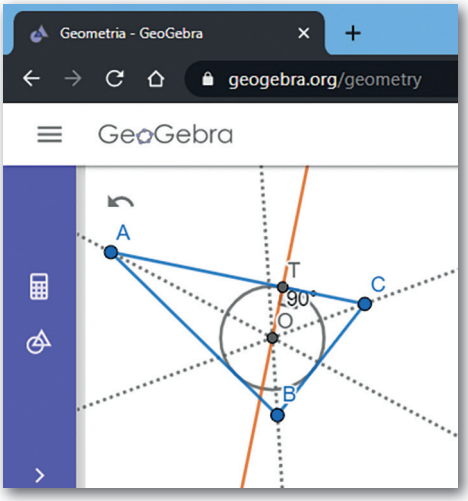
Számítógépre az operációs rendszertől függően akár többféle módon is telepíthetünk alkalmazást. Az Ubuntu esetében lehet az Ubuntu Software alkalmazásból, Windows esetén a Microsoft Store alkalmazásból telepíteni. Mindkét esetben a mobiltelefonokon megismert módon kereshetünk, nézhetjük meg az alkalmazás leírását, értékelését, a hozzá fűzött véleményeket. A telepítéshez kérheti a rendszer a rendszergazdai jogosultsággal rendelkező felhasználó hitelesítési adatait.



▶ GeoGebra telepítése Ubuntu alatt az Ubuntu Sotware áruházból

A programok egy széles köréhez a szoftverfejlesztő weboldalán keresztül férhetünk hozzá. Innen tölthetjük le a telepítőállományt. Ilyenkor nagyon figyeljünk arra, hogy megbízható oldalról töltsük le a szoftvert. Előfordul, hogy egyes fizetős szoftvereket bizonyos weboldalak ingyen kínálnak, de ebben az esetben nagy az esélye, hogy a szoftver rosszindulatú összetevőket is tartalmaz, így a gépünk feletti irányítást, vagy személyes adatainkat próbálják megszerezni az ingyenesen elérhető szoftver segítségével.

Napjainkban egyre gyakoribb, hogy bizonyos szoftverek online érhetők el. Ilyen szoftverek lehetnek akár szövegszerkesztő, képszerkesztő alkalmazások is, vagy akár a fentebb példaként többször említett GeoGebra is. Ennek használatához nem szükséges újabb szoftvert telepítenünk a gépünkre, elegendő az alkalmazás weboldalát megnyitnunk a böngészőnkben. Itt is előfordulhat, hogy egy szolgáltatásért fizetnünk kell. Ilyenkor a weboldalon belül az alkalmazás használatához csak megfelelő felhasználói azonosítóval tudunk belépni.



▶ GeoGebra online használható változata

Hasznos szolgáltatások

Az operációs rendszerek azon kívül, hogy lehetőséget biztosítanak a programok futtatására, rengeteg segédprogramot tartalmaznak, amelyek nem elengedhetetlen részei az operációs rendszernek, de a digitális eszközünk használatát megkönnyítik. Ilyen program például egy **számológép**, egy egyszerű **képszerkesztő**, egy **stopperóra**, ami nem feltétlenül szükséges, de hasznos.

Nézzünk pár példát arra, hogy mik az operációs rendszer hasznos, a mindennapi munka szempontjából nélkülözhetetlen segédprogramjai.

Védekezés a digitális kártevők ellen

Sokat hallunk a számítógépes vírusokról, férgekről, kémprogramokról, agresszív reklámprogramokról, arról, hogy ezek mennyi kellemetlenséget, problémát okozhatnak. Ezeket a programokat együttesen **rosszindulatú szoftverek**nek is szoktuk nevezni. A rosszindulatú szoftverek célja lehet a fájlok, adatok törlése, módosítása, a fájlok titkosítása annak érdekében, hogy a dekódolásért zsarolhassanak, jelszavak, bankkártya-adatok megszerzése, a megfertőzött gép használata illegális tevékenységekre, mint például a spam küldés.

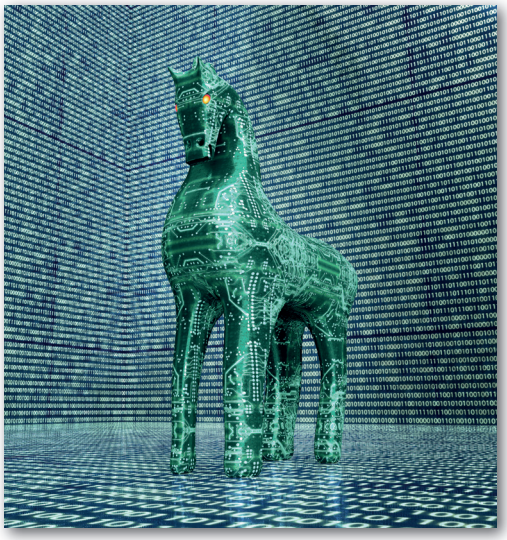
Vegyük sorra a digitális kártevők csoportjait, és azt, hogy hogyan védekezhetünk ellenük. A leggyakrabban talán a **számítógépes vírusok** elnevezéssel találkozunk. Ezek olyan programok, amelyek másik programhoz, vagy a rendszerbetöltésért felelős tárterületre írják magukat, ezzel elérve, hogy a digitális eszközünk, illetve a programok normális működésekor elinduljanak, és a rendszerben szinte tetszőleges műveletet végezhessenek. Ebből az egyik lépés az, hogy igyekeznek önmagukat újabb fájlokhoz másolni, ezzel biztosítani a szaporodást. A vírus elnevezés azért találó, mert ezek a programok is csak parazitaként, más programhoz kapcsolódva képesek szaporodni, mint a biológiai vírus egy élőlény sejtjében. Számítógépes vírus leggyakrabban nem megbízható forrásból származó programokkal jut a digitális eszközre. Ellenük a *tudatos, odafigyelő számítógéphasználói magatartás* mellett a *víruskereső és -irtó program*mal tudunk védekezni. Ez sok esetben az operációs rendszer mellé adott segédprogramként áll a rendelkezésünkre. Sokakban él az a tévhit, hogy vírusos csak a Windows lehet, ezzel szemben viszont fontos tény, hogy minden digitális eszközünk ki van téve ennek a kockázatnak, legyen az Windows, Linux, macOS rendszert futtató számítógép, vagy akár a táblagépünk, mobiltelefonunk.

► Vírustalálat

A **féreg** (angolul: worm) a vírussal szemben nem más programokhoz kapcsolódva képes terjedni, hanem ezt teljesen önállóan teszi. Általában a számítógépes hálózaton terjed úgy, hogy kihasználja a rendszerekben fellelhető programhibákat, biztonsági réseket. Védekezni vírusirtó programokkal lehet ellene, valamint azzal, hogy az operációs rendszerünket és a gépünkre telepített programokat rendszeresen frissítjük, mert a frissítések sok esetben a fejlesztők által megismert biztonsági réseket foltozzák be, ezzel is megakadályozva a rosszindulatú programok terjedését. A férgek hálózati terjedésének megakadályozásában még nagy segítséget jelentenek a *tűzfal program*ok, amik a számítógépes hálózaton keresztüli

forgalmat szűrik. A tűzfal programok is szinte kötelező segédprogramjai az operációs rendszereknek.

A **trójai program** a nevét a görög mitológiából ismert trójai falóról kapta. A trójai egy olyan program, ami másnak mutatja magát, mint ami valójában. Gyakori, hogy keresünk egy probléma megoldására egy programot az interneten, és nem megbízható forrásból, ellenőrizetlen programot telepítünk a gépünkre. A trójai program elvégzi azt a feladatot, amire beszereztük, de mellette olyan tevékenységeket is végez, amik kárt okozhatnak. Ilyen lehet például a gépen található állományok titkosítása, és ezáltal elérhetetlenné tétele. Miután egy ilyen program a tárolt adatok jelentős részét titkosította, megjelenít egy üzenetet, hogy hova milyen módon kell a „váltságdíjat” befizetni azért, hogy a feloldókódot megkapjuk. A lekövethetetlenség érdekében napjainkban kriptovalutában (például Bitcoin-ban) várják az összeget a szoftver készítői, de az, hogy fizetünk, még semmi garanciát nem jelent arra, hogy az adatainkhoz valóban hozzá fogunk utána férni. A trójai programok sok esetben nem ilyen látványosan fejtik ki hatásukat, hanem csak egy „hátsóajtót” nyitnak a gépen, azaz lehetőséget biztosítanak a szoftver készítőinek, hogy a gépünkre újabb programot juttassanak, a gépünk feletti irányítást magukhoz vehessék. Sok esetben ilyenkor a gépünk látszólag végzi a dolgát, csak kicsi lelassulást érzünk, esetleg az internetelérés sebességével leszünk elégedetlenek. Eközben a gépünk a háttérben ezerszámra küldi más felhasználók postaládáiba a kéretlen reklámleveleket, más néven spam-eket. Védekezni tudatos géphasználattal lehet, azaz ellenőrizetlen forrásból nem telepítünk semmit, valamint a vírusirtó programok nyújtanak még segítséget.



► Trójai faló

A **kémprogram**ok (angolul: spyware) olyan, főleg interneten terjedő szoftverek, melyeknek feladata felhasználói adatok megszerzése, mint például személyazonosító adatok, bankkártya-adatok, jelszavak, amelyek felhasználásával általában bűncselekményeket hajtanak végre. Ilyenek lehetnek mások nevében kötött szerződések, kötelezettségvállalások, bankszámlák megcsapolása, illetve a jelszavak felhasználásával akár nemkívánatos üzenetek küldése, a felhasználó ismerősei elérhetőségének megszerzése kéretlen reklámok továbbítása céljából.

A **rosszindulatú szoftverek elleni védekezés**nek több módja van, amelyek mindegyikét érdemes megtenni. Az elsődleges a tudatos géphasználat. *Megbízhatatlan forrásból nem telepítünk szoftvert*, hiába csábít, hogy egy egyébként drága szoftvert majd ingyen használhatunk. A számítógépes kártevők egy része e-mail mellékletként érkezhet meg hozzánk akár egy ismerősünk e-mail címéről. Ezért legyünk mindig elővigyázatosak a mellékletek megnyitásakor. A digitális eszközünkön – legyen az számítógép vagy akár mobiltelefon – legyen *víruspajzs program*, ami minden állományt megvizsgál használat előtt. Mivel a kártevők újabb változatai naponta jelennek meg, fontos, hogy a vírusirtó program naprakész

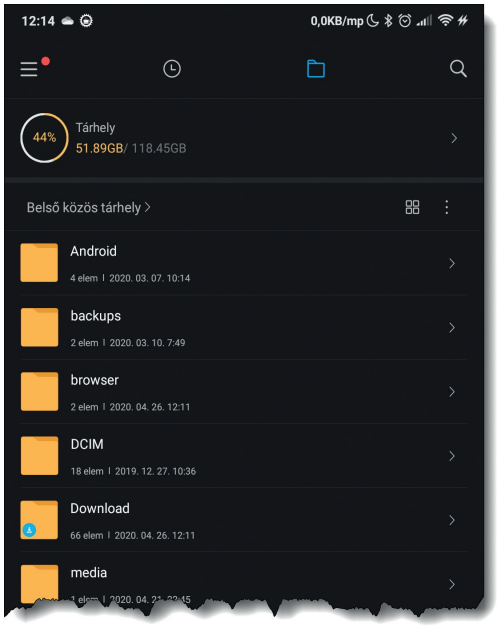
legyen, ezért rendszeresen töltsük le a *vírusdefiníciós adatbázis*át. Ezt általában a vírusirtók automatikusan megteszik, csak engedélyeznünk kell számukra ennek végrehajtását. A rosszindulatú programok sokszor a programokban található biztonsági hibákat használják ki. A szoftvergyártók az ismertté vált hibákat rendszeresen javítják, és *frissítések kiadásával* juttatják el a felhasználókhoz. Ezeket a frissítéseket a megjelenésük után a lehető leghamarabb telepítsük eszközeinkre! Amennyiben az operációs rendszerünkön van *szoftveres tűzfal*, azt tartsuk bekapcsolva. Ha valamilyen program a tűzfal miatt nem működik rendesen, akkor ne a tűzfal kikapcsolása legyen a megoldás, hanem keressük meg, hogy milyen beállítások mellett lesz a programunk működőképes úgy, hogy közben a rendszerünk védett marad.

Készüljünk fel arra, hogy számítógépünk rosszindulatú támadás áldozatává válik. Ha más megoldás nincs, akkor az eszközünkön vissza kell állítani az operációs rendszer gyári állapotát. Ezzel mind a feltelepített programjainkat, mind az állományainkat (képeinket, dokumentumainkat) elveszítjük. Ezért készítsünk rendszeresen *biztonsági mentés*t adatainkról, programjainkról, és ezt tartsuk lehetőleg fizikailag külön az eszközünktől. Lehet egy külső tárolón, vagy akár egy felhős tárterületen. Ezt rendszeresen végezzük el, de ehhez akár beállíthatunk automatizált mentést is.

► Biztonsági mentés külső meghajtóra

Fájlkezelés

A digitális eszközeinken állományokat tárolunk, amelyek kezelését egy **fájlkezelő** megkönnyíti. Itt mappákba rendezve láthatjuk az állományokat. Magunk is hozhatunk létre új mappákat, és áthelyezhetjük állományainkat, hogy később könnyen megtaláljuk azokat. Például e könyv készítése közben a könyvbe kerülő képállományokat fejezetenként külön-külön mappákba gyűjtöttük, hogy a kiadványszerkesztéssel foglalkozó kollégák számára jól kezelhető legyen. Érdemes a különböző projektek anyagait, azon belül a munkaanyagokat, részben vagy egészben feldolgozott fájlokat elkülönítetten tárolni. Így amikor a végeredményt be kell adni, akkor könnyen össze tudjuk gyűjteni a beadandó állományokat, ha viszont még újabb módosítási igények merülnek fel, akkor vissza tudunk nyúlni az eredeti források-



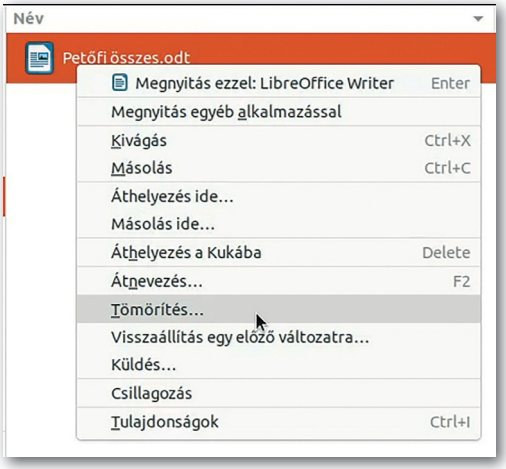
► Fájlkezelő Android alatt

hoz. Amennyiben nem csak a saját eszközünkön szeretnénk tárolni az állományokat, esetleg meg szeretnénk osztani másokkal is az iskolában, akkor szükségünk lesz egy fájlszerver elérésére. Sok iskolában a helyben üzemeltetett szerveren lehet tárolni a tanuláshoz kapcsolódó anyagokat. Ilyenkor mindenkinek egyedi felhasználói azonosítóval és jelszóval kell rendelkeznie. Ezzel biztosítható, hogy mindenki a saját állományaihoz, mappáihoz hozzáférjen, a többiekét ne lássák, viszont legyenek olyan tárterületek, amit a diákok egymással, a tanárral is közösen használhatnak. A mai operációs rendszerek a helyi fájlkezelés mellett a hálózati fájlkezelést is alapszolgáltatásként biztosítják, általában a hálózaton tárolt adatok elérése nem, vagy csak kis mértékben tér el a helyi tárolásnál megszokottól.

Tömörítés

Rendszeresen előfordul, hogy elkészült munkánkat másokkal kell megosztanunk. Ilyenkor probléma lehet az állomány mérete, vagy esetleg az, hogy egy mappán belül sok állományt, esetleg mappát kellene átadnunk. Ebben az esetben segíthet a tömörítés. Elsőként gondoljuk végig, mit is jelent a tömörítés. Amikor adatokat akarunk tárolni, akkor az elfoglal valamekkora helyet a memóriában vagy a háttértáron. Hogyan lehet ezt tömörebben, kisebb helyen tárolni? Ehhez fontos tudnunk, hogy hogyan tároljuk az adatokat. A számítógépes grafika fejezetben a pixelgrafikus képszerkesztésnél volt már pár szó a tömörítésről. A tömöríthetőséget grafikus példával könnyű szemléltetni.

Képzeljünk el egy olyan rajzot, ahol a kép felső fele kék, alsó fele zöld színű. Ekkor tárolhatjuk minden egyes képpont színét egy számértékkel. Így a kép méretével egyenes arányban nő a tárolandó adat mennyisége. Ezzel szemben tárolhatnánk az egymás utáni azonos színű képpontok esetében azt, hogy hány darab milyen színű képpont következik. A példánkat megnézve látható, hogy ezzel a tárolással az eredeti tárolási mérethez képest jelentősen kisebb tárterületre lesz szükségünk, de szükség esetén képpontról képpontra vissza tudjuk állítani az eredeti képet. A kép mérete kisebb lesz, cserébe számítási kapacitásokat igényel a folyamat. Ez a tárolási mód veszteségmentes, azaz teljesen pontosan visszaállítható az eredeti adathalmaz. A példa alapján elképzelhetjük, hogy nem csak képeken, hanem más állományokon is van lehetőségünk ismétlődő, vagy gyakran megjelenő mintázatokat találni, amik rövidített formában helyettesíthetők.



▶ Tömörítés – Ubuntu fájlkezelő

Amikor egy vagy több állományt szeretnénk másnak átadni, és ezért tömörítjük, akkor egyértelműen veszteségmentes tömörítésre van szükségünk. Az operációs rendszerbe integrált, vagy külön programként megjelenő fájltömörítők így működnek. A leggyakrabban használt ilyen tömörítési formátum a **ZIP**. Az operációs rendszerek fájlkezelő alkalmazásaiban általában a fájlokhoz vagy mappákhoz kapcsolódóan előhívható helyi menüben is megjelenik ez a lehetőség, ahogy az a képen is látható. Amennyiben több állományt vagy mappát jelölünk ki, akkor a tömörítés eredményeként egy darab tömörített állományt ka-

punk. Ez akkor is jól jöhet, ha sok állományt kellene továbbítanunk, mert így egy csomagban tehetjük ezt meg. Ebben az esetben a tömörített állományban a mappaszerkezet is tárolódik, azaz kibontás után ugyanabban a struktúrában lesznek elérhetőek az állományok, mint ahogy eredetileg a tömörítés előtt voltak. Fontos tudnunk a veszteségmentes tömörítés kapcsán, hogy egy már tömörített állomány tömörítésével már csak nagyon korlátozott mértékben tudunk helyet megtakarítani, ha egyáltalán lehet. Amennyiben olyan állományt próbálunk tömöríteni, ami már eleve (veszteséges vagy veszteségmentes) tömörített formátum, akkor az eredeti mérethez nagyon közeli méretet kapunk eredményül. Tömörített formátumok például a képek esetén a JPG, a PNG, hangok esetén az MP3, a FLAC, videók esetén az MP4, a MOV. A Microsoft Word által használt DOCX, és a LibreOffice Writer által használt ODT is már tömörített formátum, ráadásul a ZIP állományok létrehozásához használt algoritmussal tömörítettek.

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 melyik 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 megnövekvő költségei lesznek, de nagyobb forgalomból várhatóan nagyobb bevétel is következik. Mivel a szerverek üzemeltetését erre szakosodott szervezet végzi, a szolgáltatás *rendelkezésre állás*a is magasabb, kevesebb rendszerhibából adódó időkieséssel kell számolni. Ez átlagosan éves szinten legfeljebb néhány perc. Egy szerverközpontban a feladatok a nagyszámú gép között jól elosztható, így egy cég igényeinek megfelelő szolgáltatás sokkal költséghatékonyabb, mintha azt saját eszközökkel, saját alkalmazásban lévő rendszergazdával kellene megoldania.

A felhőszolgáltatásoknak az egyik nagy előnye, 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 első a kommunikáció!

Lehetőségünk van *levelezőszolgáltatás*t igénybe venni. Sokak által használt a Gmail, az Outlook, a Microsoft365, a ProtonMail, a Yahoo Mail.

A levelezés mellett a kapcsolattartás rövid, azonnali szöveges üzenetekben is zajlik, ez a *csevegés* (angolul chat). Gyakran használt csevegő szolgáltatás a Messenger, a Viber, a Skype, a WhatsApp. Ha hang alapon szeretnénk kommunikálni, akkor azt általában a csevegőprogramokkal is megtehetjük.

*Videóhívás*okra is lehetőséget szoktak biztosítani a csevegőprogramok, de vannak olyan szolgáltatások, amik kifejezetten ebben erősek. Ilyen például a Microsoft Teams, a Webex Meetings, a Zoom, a Google Meet.

▶ Videókonferencia

Ha állományokat szeretnénk tárolni, másokkal megosztani, közösen használni, akkor a felhős tárhelyszolgáltatásokat vehetjük igénybe, például a Microsoft OneDrive, a Google Drive, a Dropbox. Ezeknél lehetőségünk van automatikus szinkronizálás beállítására. Ha beállítjuk, akkor a telefonunkon készített fényképek automatikusan a felhős tárterületünkre másolódnak, így nem kell külön gondoskodnunk azok biztonsági másolatáról. Ilyenkor beállíthatjuk, hogy a mobilhálózati adatforgalom esetén is engedélyezett legyen a szinkronizálás, ami külön költségeket jelenthet, vagy csak a 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 azokat másokkal megosztani. A megosztásnál meghatározhatjuk, hogy azt mindenki elérheti-e, vagy csak az általunk engedélyezett személyek. Beállíthatjuk, hogy csak megtekinteni tudják, vagy lehetőségük legyen módosítani is azokat. Van olyan szolgáltatás, ahol még időkorlátot is be lehet állítani, például 3 napig elérhető a linken, utána már nem lesz megosztott. Ha egy projektfeladat kapcsán a mobiltelefonunkkal készítünk egy videót vagy fényképeket, akkor azokat a telefonunkról a felhős tárterületre másolhatjuk, majd azt a csapatunk többi tagjával megoszthatjuk. Ha egy megosztott mappába újabb állományokat teszünk, akkor azt a többiek azonnal elérik.

Ha *dokumentumokat* szeretnénk *szerkeszteni* a többiekkel egyidőben, akkor használhatjuk például a Google Dokumentumok vagy a Microsoft365 szolgáltatást.

A felsorolt felhőszolgáltatásokban közös, hogy igénybevételük *eszközfüggetlen*, azaz laptopon, táblagépen, okostelefonon egyaránt elérhetőek, 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, amit 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 ennek segítségével nem tudnak a nevünkben bejelentkezni, ha a telefonunk mindeközben nálunk van.

Feladatok, kérdések

1. Nézzük meg, hogy mobiltelefonunkat be tudjuk-e úgy állítani, hogy azt kölcsönadva csak a telefonálás szolgáltatást tudják használni, az üzeneteinket, fényképeinket ne tudják megnézni, a telepített programokat ne tudják elindítani, nevünkben ne tudjanak üzenetet küldeni!
2. Hogyan lehet egy mobiltelefonra letöltött fájlt másik mappába áthelyezni?
3. Milyen módokon tudunk egy mobiltelefonnal készített fényképet a teremben lévő osztálytársunknak átküldeni? Milyen lehetőségeink vannak nagyobb távolság esetén, például, ha iskolaidő után már otthon vagyunk, és egy közös projekthez szükséges megosztanunk egy általunk készített képet?
4. Milyen műveleteket lehet végezni a szövegszerkesztő programban a funkcióbillentyűk segítségével? Nézzük meg a SHIFT, a CTRL és az ALT billentyűkkel együttes használatot is!
5. Hogyan lehet egy tömörített állományt jelszóval védetté tenni?
6. Egy képeket és szöveget is tartalmazó DOCX vagy ODT állomány kiterjesztését írjuk át ZIP-re, majd bontsuk ki! A megkapott könyvtárak és állományok között keressük meg a szövegszerkesztővel elkészített állományban lévő szöveget és képeket!
7. Milyen felhőszolgáltatásokat érdemes használni egy 3-4 fős csapatban megoldandó projekt kapcsán? A különböző projektek a használandó eszközökben is eltérhetnek, így érdemes többféle projekt esetében is végiggondolni a választ.