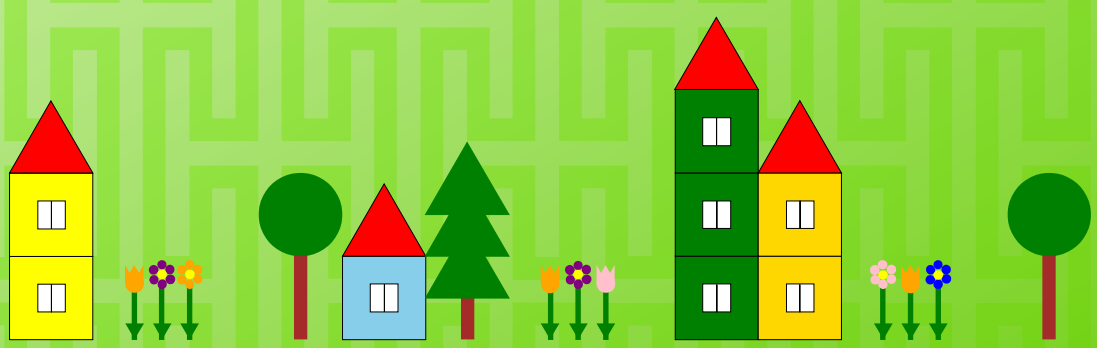
E-közigazgatási Szabad
Szoftver Kompetencia Központ

Lakó Viktória

LibreLogo
oktatási segédanyag

A teknőcgrafikától a programozásérettségiig



ÚJ SZÉCHENYI TERV

Lakó Viktória:
LibreLogo oktatási segédanyag – A teknőcgrafikától a programozásérettségiig
Elektronikus kiadás
Lektorálta: Torma Hajnalka és Németh László
Kiadó: E-közigazgatási Szabad Szoftver Kompetencia Központ, 2013
ISBN 978-963-08-8253-8

Változat: 2014. március 28. A könyv és ábraanyaga LibreOffice 4-ben készült.
A könyv honlapja: http://szabadszoftver.kormany.hu/sajat-oktatasi-anyagok/

Tartalomjegyzék

Előszó

Szeretnék köszönetet mondani Németh Lászlónak a folyamatos konzultációért, feladatötletekért és megoldásvariációkért, valamint a mű szakmai lektorálásáért. A könyv megírásához nélkülözhetetlen volt Laci korábban megjelent LibreLogo kézikönyve, amelynek segítségével megismerhettem ezt a nagyszerű programot.

Köszönet illeti Mahler Attilát és Cseppentő Árpádot, akik a nyelvi hibák javításával és építő kritikáikkal vettek részt a könyv végleges formájának kialakításában.

Lakó Viktória

Bevezető

Kedves Olvasó!

A könyv, amelyet kezedben tartasz, egy diákoknak és tanároknak szánt tanulási, tanítási segédlet. Alapvetően a LibreOffice LibreLogo nevű kiegészítő programjának oktatásban való felhasználásához nyújt segítséget, de más Logo-szerű nyelvek elsajátításához is jó alapot ad. A könyv először bemutatja a teknőcgrafika alapjait, majd fokozatosan egyre komolyabb programozási eszközöket, módszereket is tárgyal.

A LibreLogo egy gyerekeknek szánt programozási környezet, amelyet Németh László fejleszt. Az első verziókat külön kellett telepíteni a LibreOffice (vagy akár az OpenOffice.org) kiegészítőjeként, de a LibreOffice 4 eleve tartalmazza a Logo kiegészítőt. A LibreLogo egyedülálló módon vektorgrafikus ábrákat készít, amelyek bármilyen nagyításban szépen mutatnak, és a LibreOffice beépített alakzataihoz hasonlóan szabadon formázhatók a rajzolást követően. Python nyelven íródott, és képes használni ennek a fejlett programozási nyelvnek a program- és adatszerkezeteit, ezáltal a LibreLogóval akár komolyabb (nem csupán teknőcgrafikus) feladatokat is megoldhatunk. Mindezt magyar nyelven. Ezáltal ez a programnyelv kiválóan alkalmas a programozás bevezetésére a közoktatásban.

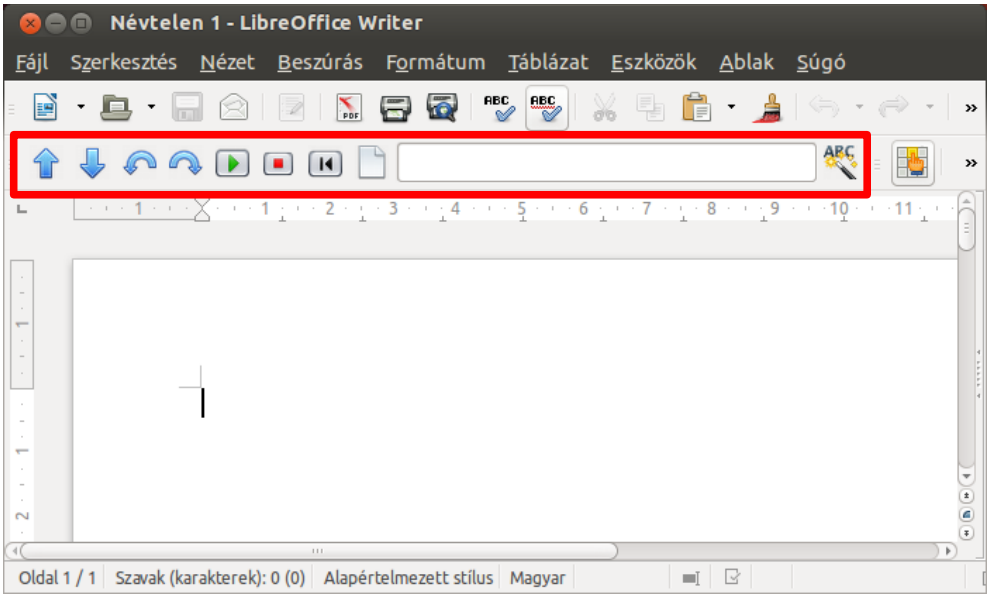
A könyvben a megoldott mintapéldák mellett további feladatok, gondolkodási irányok is fellelhetők. A feladatok megoldását egy külön megoldáskötetben teszszük közzé elektronikus formában.

Alapparancsok (teknőcgrafikai alapok)

A LibreLogo telepítése, elindítása

Mielőtt megismerkedünk a teknőcgrafikai alapokkal, elengedhetetlen pár szót ejteni a LibreLogóról. A LibreLogo a LibreOffice programcsomag szövegszerkesztőjének, a Writernek a kiegészítője. Tehát használatához szükséges a LibreOffice programcsomag telepítése. Ajánljuk, hogy a LibreOffice legfrissebb stabil verzióját telepítsd, hiszen ez már eleve tartalmazza a LibreLogót, és ebben javításra kerültek a korábbi verziók esetleges hibái. A LibreOffice telepítőjét ingyenesen letöltheted a http://libreoffice.hu/ weboldalról, itt segítséget is találsz a telepítéshez.

Miután rendelkezünk telepített LibreOffice-szal, indítsuk el a Writer szövegszerkesztőt! Az elindítás után egy ehhez hasonló kép fogad:



A fenti képen pirossal keretezve látható a Logo eszköztár, ami elképzelhető, hogy nálad egy picit máshol van, esetleg nem is jelenik meg. Ez utóbbi esetben a Nézet–Eszköztárak...–Logo menü segítségével tudod megjeleníteni a Logo eszközöket.

A LibreLogo eszköztár

Mielőtt ismertetném a Logo eszköztár ikonjait, ejtek néhány szót magáról a Logo nyelvről, illetve a teknőcgrafikáról. A Logo nyelv eredete egészen 1967-re nyúlik vissza. Wallace Feurzeig és Seymour Papert fejlesztette ki a cambridge-i BBN[1] kutatóintézetben ezt a gyerekek oktatására alkalmas programozási környezetet. A program első változata funkcionális szövegfeldolgozásra volt alkalmas, a teknőc csak egy későbbi fejlesztésben jelent meg.

Az első teknőc, amelyet Irvingnek neveztek, valódi robot volt, amelyet rádióval lehetett irányítani. Értette az előre, hátra, jobbra, balra és csenget parancsokat. A grafikus felületek megjelenésével a fizikai robotot grafikai programozási környezetek tucatjai váltották fel, amelyek ezt a teknőcirányító filozófiát vették alapul. Ilyen például a Magyarországon is ismert Comenius Logo és az Imagine Logo.

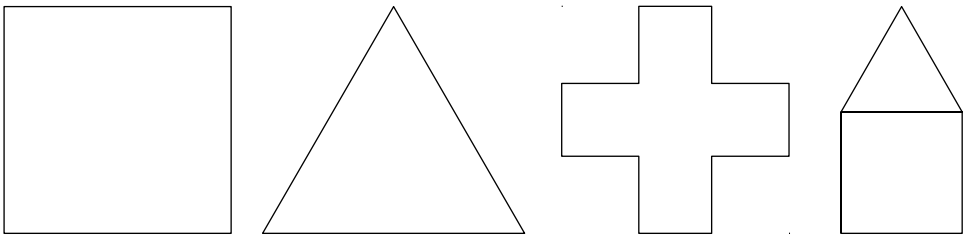
A LibreLogóban szintén egy teknőcöt fogunk irányítani különböző parancsokkal. A teknőcnek egy toll van a hasa alatt, amellyel nyomot hagy maga után a papíron, tehát a mozgatásával különböző ábrákat rajzoltathatunk vele. A kérdés már csak az, hogyan csaljuk elő a teknőcöt? Ehhez elegendő a Logo eszköztár bármelyik ikonját megnyomnunk, rögtön megjelenik a teknőcünk a lap közepén. Mit is csinál a teknőc az eszköztár különböző ikonjainak hatására?

Ikon	Mely utasítást hajtja végre?	Mi az utasítás hatása?
	előre 10	A teknőc előre megy (a feje irányába) 10 pontnyit.
	hátra 10	A teknőc hátra megy (a farka irányába) 10 pontnyit.
	balra 15	A teknőc balra fordul 15°-ot.
	jobbra 15	A teknőc jobbra fordul 15°-ot.
		A dokumentumba írt programkód végrehajtását eredményezi. Ha van kijelölt szöveg, csak azt hajtja végre. A LibreOffice 4.3-tól üres dokumentumban egy példaprogramot indít el.

[1] A tanácsadó és kutatóintézetet az MIT három professzora alapította, Richard Bolt, Leo Beranek és Robert Newman, a nevük kezdőbetűiből alkotott rövidítés a BBN.

		A futó program leállását eredményezi.
	haza	A teknőc a kiinduló pozícióba kerül (lap közepére) és visszanyeri a kezdőbeállításait (tollállapotait).
	törölkép	Letörli a lapról a rajzolt alakzatokat (a szöveget nem).
		Parancsokat formáz, és ha szükséges, a más nyelvű Logo programot lefordítja a dokumentum nyelvére. A LibreOffice 4.3-tól kétoldalas programszerkesztő felületet állít be a megfelelő nagyítással és oldaltöréssel.

Ismerkedésképp rajzold meg az eszköztár segítségével a következő alakzatokat:



A fenti alakzatok megrajzolásához elegendő az előre, hátra, jobbra, balra ikonokat használnod. Ha elrontottál valamit, a törölkép és haza gombokkal tudod újrakezdeni a munkádat. Most már megtapasztaltad, hogyan mozog a teknőc a képernyőn, hogyan tudod őt irányítani. Azonban az is biztosan felmerült benned, hogy bonyolult rajzok elkészítése ezzel a módszerrel nem túl kényelmes. Ha elrontunk valamit, akkor az egész munkát újra kell kezdeni. Ésszerű ötletnek tűnik ez után, hogy jegyzeteljük magunknak a lépéseket. A fenti rajzokhoz a következő jegyzeteket készíthetjük (E – előre 10, J – jobbra 15, B – balra 15):

- EEEJJJJJJEEEJJJJJJEEEJJJJJJEEEJJJJJJ vagy 3E 6J 3E 6J 3E 6J 3E 6J;
- JJ EEEJJJJEEEJJJJEEEJJJJ vagy 2J 3E 4J 3E 4J 3E 4J;
- EBBBBBB E JJJJJJEJJJJJJEBBBBBB E JJJJJJEJJJJJJEBBBBBB E JJJJJJEJJJJJJEBBBBBB E JJJJJJEJJJJJ vagy E 6B E 6J E 6J E 6B E 6J E 6J E 6B E 6J E 6J E 6B E 6J E 6J;
- 3E 6J 3E 6J 3E 6J 3E 6J 3E 2J 3E 8 J 3E 8 J 3E 8 J.

A jegyzetkészítéssel sokat könnyítettünk a dolgunkon, de még mindig elég kényelmetlen így a rajzolás, valamint biztosan észrevetted, hogy két ikont, az *Indí-*

tást és a *Leállítást* még nem használtuk. Valójában most fog kezdődni az igazi rajzolás!

A teknőcöt nem csupán az eszköztár gombjaival, hanem parancsszavakkal is lehet irányítani. Néhány parancsszót már most is tudsz: előre, hátra, jobbra, balra, törölkép, haza. A parancsszavakat elegendő begépelned a teknőc alatti papírra (hogy írni tudj, kattints az egérrel a lapra).

Írd be az előre utasítást, majd kattints az Indítás gombra! Mit látsz? Erre számítottál? Nem, ugye? A program azt írja, hogy *„Hiba (1. sor)”*. Rájöttél, mi okozza a hibát? A hibát az okozza, hogy a teknőc, bár azt tudja, hogy azt várod tőle, hogy haladjon előre, azt nem, hogy mennyit. Javítsd a kódot a következőre:

```
előre 100
```

Ezután ismét kattints az Indítás gombra! Ezt már érti a teknőc, és előre halad 100 pontnyit (a pont egy tipográfiai[2] mértékegység, amely 2,54/72 cm ≈ 0,03528 cm-nek felel meg). Az előre utasítást tehát mindenképp egy számnak kell követnie, ezt a számot hívjuk az előre utasítás *paraméter*ének. Az utasítás és paramétere közé mindig teszünk egy darab szóközt. Ezt a paramétert nem csak pontban, hanem más mértékegységgel is meg tudjuk adni, például cm-ben, vagy mm-ben.

```
előre 2cm előre 35mm
```

A kód hatására a teknőc először 2 cm-t, azután 35 mm-t tesz meg a képernyőn, azaz összesen 5,5 cm-t[3]. Hasonlóan működik a hátra utasítás is, csak ekkor a teknőc nem a feje, hanem a farka irányába mozog a megadott távolságra. A mértékegység és a szám közé nem teszünk szóközt.

Mit gondolsz, mit csinál a teknőc a balra 90 parancs hatására? Bizony, a balra utasításnak is van paramétere, hiszen nem elég annyit mondanunk, hogy balra forogjon a teknőc, azt is meg kell mondanunk, hogy mennyit. Ha kipróbáltad a balra 90 utasítást, azt tapasztalod, hogy a teknőc 90°-ot fordult balra. Tehát, ha más mértékegységet nem adunk meg, akkor a forgást fokokban mérjük. Természetesen a fent elmondottak a jobbra utasításra is érvényesek, csak nem balra fordul a teknőc, hanem jobbra.

[2] A tipográfia a nyomtatott szövegek esztétikus megjelenítésével foglalkozó szakma és művészeti ág.

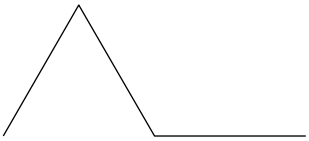
[3] Tizedes törteket is meg tudunk adni az utasítás paramétereként, a LibreLogóban nemcsak tizedespontot, hanem a magyar nyelvű programok esetében tizedesvesszőt is használhatunk, pl. előre 5,5cm.

Megjegyzés: az alapértelmezett pont és a szög mértékegységek megadhatók bővebb alakban is (a későbbiekben egy-két példától eltekintve nem ezeket, hanem az egyszerű, csak számokat tartalmazó alakokat fogjuk használni):

előre 100pt
balra 90°

Most töröld le a képernyőt, és küldd haza a teknőcöt, majd írd be a következő parancsokat:

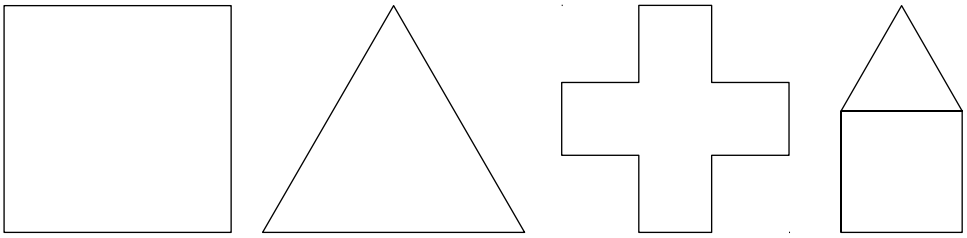
jobbra 30
előre 2cm jobbra 120 előre 2cm
balra 60 előre 2cm



Ha nem gépeltél el semmit, akkor a teknőc a kód melletti ábrát rajzolja meg. A fenti példából kiderül, hogy az utasításokat a teknőc szépen sorban egymás után hajtja végre, balról jobbra felülről lefelé olvasva. Az utasításokat írhatjuk külön sorba, vagy egymás után szóközzel elválasztva. Javaslom, hogy rövid rajzolási egységek után kezdj új sort, ezáltal később könnyebben javítható a kódod.

Most, hogy már megismerkedtünk a négy alapvető mozgató utasítással, térjünk vissza a négy ábránkhoz:

A készített terveket is felhasználva rajzoljuk meg az ábrákat, csak a most tanult négy utasítás segítségével. A 4. rajz tervét fogjuk most átírni, a másik három rajz elkészítését rád bízom!



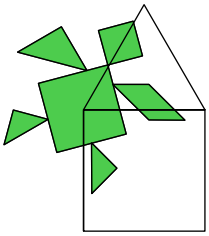
Vegyük először a tervet: 3E 6J 3E 6J 3E 6J 3E 6J 3E 2J 3E 8J 3E 8J 3E 8J.

Itt a E az előre 10 , a J a jobbra 15 utasításoknak felel meg, a betűk előtti számok, pedig azt jelzik, hogy az adott gombot hányszor kell lenyomnunk, tehát az

- 3E – előre 30 (3*10) és a
- 6J – jobbra 90 (6*15)

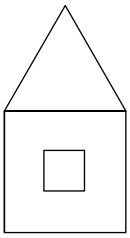
így fordíthatók le a teknőc nyelvére. Ez alapján a házikó rajzot a következő utasításokkal lehet megrajzolni (a teknőc a ház bal alsó sarkából indul észak felé, és a ház bal felső sarkán áll meg az utasítások végrehajtása után, a képen látható irányba fordulva):

előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 30
előre 30 jobbra 120
előre 30 jobbra 120
előre 30 jobbra 120



Hogy ne kelljen mindig ennyit gépelni, szinte minden Logo utasításnak van rövid változata, ezeket minden új utasításkor leírom, de a könnyebb érthetőség kedvéért a könyvben mindenhol az utasítások teljes alakját használom.

Az elkészült házikóra szeretnénk ablakot rajzolni az alábbi módon:

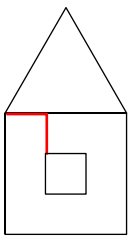


Hogyan jutunk be a teknőccel a házikó belsejébe anélkül, hogy vonalat húznánk? Ehhez két új parancsot fogunk megtanulni:

- tollatfel – felemeli a teknőc tollát, ezután a parancs után a teknőcöt szabadon mozgathatjuk, nem húz vonalat. Rövid változat: tf.
- tollatle – ha a toll felemelése után újból rajzolni szeretnénk, akkor ezzel az utasítással tudjuk lerakni a tollat, hogy a teknőc ismét nyomot hagyjon a papíron. Rövid változat: tl.

A fenti két utasításnak nincsenek paraméterei. Próbáljuk ki, hogyan tudjuk segítségükkel megrajzolni a házikóra az ablakot. Ehhez a következő pár sort kell az előbbi kód végére írni:

tollatfel
jobbra 60
előre 10 jobbra 90 előre 10
tollatle
előre 10 balra 90
előre 10 balra 90
előre 10 balra 90
előre 10 balra 90



Itt a házikó megrajzolása után felemeljük a tollat, hogy az ablak jobb felső sarkához jussunk anélkül, hogy vonalat húznánk. Fejben kellett tartanunk, hogy hol fejeztük be a rajzolást, és hogy merre néztünk. Felemelt tollal a fenti képen pirossal jelölt útvonalat jártuk be.

Ez a szép házikó most már megérett rá, hogy elmentsük magunknak! Kattints a *Fájl–Mentés* parancsra! A megjelenő ablakban adj a programnak nevet, pl. házikó, és mentsd el egy olyan mappába, ahol később is megtalálod!

Ez már egy egész remek kis házikó, de mennyire szépen mutatna, ha ki is színeznénk! Természetesen ez is lehetséges, méghozzá kétféle módon:

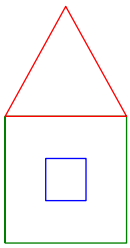
- a körvonalat rajzoljuk meg különböző színekkel, vagy
- a fehér területeket színezzük be.

Először az első módszerrel fogunk megismerkedni. Ehhez rajzolás közben kell a teknőc tollának színét változtatnunk a következő módon: tollszín „kék”. Rövid változat: tsz „kék”.

A tollszínt változtató parancs, mint gondolom, te is kitaláltad, szintén egy paraméteres utasítás. Paraméterként meg kell adnunk, hogy milyen színűre változtassa teknőcünk a tollát. A színt a nevének idézőjelek közötti begépelésével adhatjuk meg. Hogy mely színeket ismeri a teknőc, arról a könyv végén találsz egy táblázatot.

A tollszín „kék” utasítás után a teknőc kék színű vonalat fog húzni egészen addig, míg a tollszínét valamely másik színre nem változtatjuk. A toll színét ezért a teknőc állapotának hívjuk, a teknőc tollának aktuális színét a teknőc körvonalából tudjuk leolvasni. Ennek ismeretében módosítjuk a korábbi kódunkat, hogy színes házikót készítsünk:

```
tollszín „zöld”
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 30
tollszín „piros”
előre 30 jobbra 120
előre 30 jobbra 120
előre 30 jobbra 120
tollatfel
jobbra 60
előre 10 jobbra 90 előre 10
```

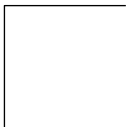


```
tollatle
tollszín „kék”
előre 10 balra 90
előre 10 balra 90
előre 10 balra 90
előre 10 balra 90
```

A házikó belsejének kiszínezése egy picit összetettebb dolog lesz. Két új utasítást is meg kell tanulnunk hozzá.

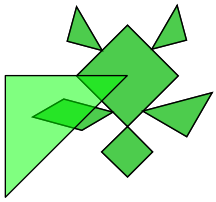
Először ismerkedjünk meg a tölt paranccsal. Írd be a következő kódot:

```
előre 30 jobbra 90
előre 30 jobbra 90
```



A fenti két sor a mellette levő töröttvonalat rajzolja meg, a rajzolás után a teknőc az ábra jobb felső sarkában van és lefelé néz. Figyeljük meg, hogy mi történik, ha ezt kiegészítjük a tölt utasítással:

```
előre 30 jobbra 90
előre 30 jobbra 90
tölt
```

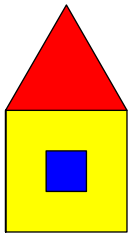


Láthatod, hogy a vonal bezárul, mégpedig oly módon, hogy a töröttvonal végét összeköti az elejével. Mindeközben a teknőc pozíciója nem változik. Az így kapott zárt terület pedig a teknőc színével megegyező színt kap, ami alapesetben egy 50% átlátszóságú zöld szín.

A teknőc töltőszínét is meg tudjuk változtatni, például pirosra a töltőszín „piros” utasítással. Rövid változat: tlsz „piros”.

Készítsük hát el a kitöltött házikót is (most a körvonal színét feketén hagyjuk):

```
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 90
előre 30 jobbra 30
töltőszín „sárga” tölt
előre 30 jobbra 120
```

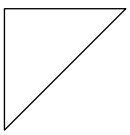


előre 30 jobbra 120
előre 30 jobbra 120
töltőszín „piros” tölt
tollatfel
jobbra 60
előre 10 jobbra 90 előre 10
tollatle
előre 10 balra 90
előre 10 balra 90
előre 10 balra 90
előre 10 balra 90
töltőszín „kék” tölt

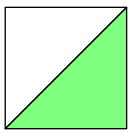
A példában a tölt utasítást mindig közvetlenül előzi meg a töltőszín beállítása. Bár a két parancs egymástól függetlenül is működik, mégis azt javaslom, hogy használd őket egymás után, hogy mindig tudd, milyen színnel színezed ki a rajzod.

Előfordul, hogy nem szeretnénk minden alakzatot kitölteni, ekkor szükség van egy olyan utasításra, amellyel új alakzat rajzolásba kezdhetünk. Ezáltal meg tudjuk különböztetni, mely ábrákra vonatkozik a kitöltés. Erre szolgál a zár utasítás, amely lezárja az addig elkészült alakzatot kitöltés nélkül.

előre 30 jobbra 90
előre 30 jobbra 90
zár



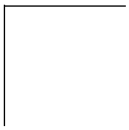
Így elkészíthetjük a következő rajzot:



előre 30 jobbra 90
előre 30 jobbra 90
zár
előre 30 jobbra 90
előre 30 jobbra 90
tölt

Amennyiben nem szeretnénk a töröttvonal elejét és végét összekötni, akkor az ábrát szakaszonként le kell zárni, így minden szakasz egy-egy külön alakzat lesz:

előre 30 jobbra 90 **zár**
előre 30 jobbra 90 **zár**



Most már ismered az alapvető parancsokat, gyakorlásképp oldd meg a fejezet végén található feladatokat! Ne felejtsd el a munkáidat szép sorban elmenteni, hogy később is meglegyenek!

Kiegészítő anyag: Színek RGB kódja

A 157. oldalon lévő táblázat tartalmazza azt a 16 színt, amelyekre a nevükkel hivatkozhatunk. Azonban a teknőc ennél jóval több színt ismer. A színeket a piros, zöld és a kék összeadó színkeverésével készíti a teknőc (ahogy a számítógépek világában máshol is csinálják). Innen is jön a most bemutatandó színkód elnevezése, ugyanis angolul a piros szín *red*, a zöld szín *green*, a kék szín pedig *blue*, ezen három szín kezdőbetűjét összeolvasva kapjuk az RGB rövidítést. A színek RGB kódja egy három számból álló lista, mindhárom szám 0 és 255 közé esik. Ezek a számok mutatják, hogy milyen a szín vörös, zöld és kék színösszetevője. Így a piros szín RGB kódja a [255, 0, 0], a zöldé a [0, 255, 0], a kéké pedig a [0, 0, 255]. A fekete szín a [0, 0, 0] kódnak, míg a fehér a [255, 255, 255] kódnak felel meg. Kísérletezz bátran különböző kódokkal, készíts saját színeket a már ismert színbeállító utasítások használatával:

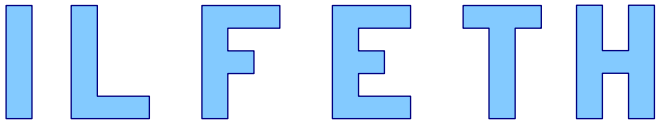
tollszín [45, 88, 200]
töltőszín [255, 200, 100]

A LibreLogóban az RGB értékek mellett van még egy negyedik „színösszetevő” is, ez az átlátszóság. Teljes átlátszósághoz rendelkezésre áll a „láthatatlan” színérték, amelynek segítségével olyan alakzatokat rajzolhatunk, amelynek nincs (látható) körvonala. Ha az alapértelmezett kitöltőszínhez hasonló félig áttetsző kitöltést szeretnénk, akkor ezt az RGB színlista negyedik (úgy nevezett „alfa”) értékkel való megtoldásával adhatjuk meg, ahol a 0 jelenti az átlátszatlanságot, a 255 pedig a teljes átlátszóságot.

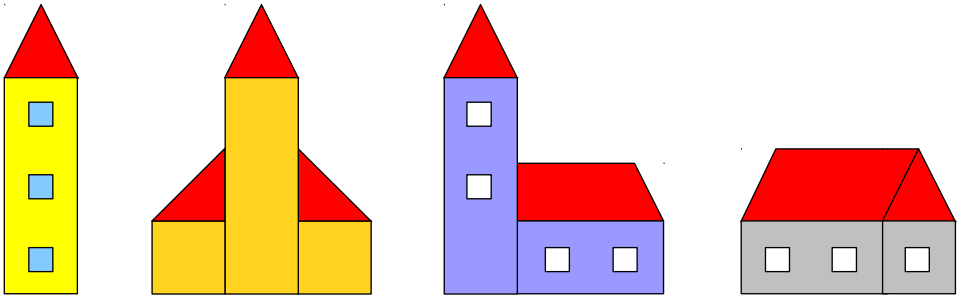
tollszín „láthatatlan”
töltőszín [0, 0, 0, 128]

Feladatok

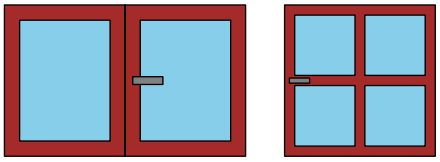
1) Készítsd el az alábbi betűket megrajzoló programokat! Készíts tervet: rajzold meg a betűket egy négyzetrácsos papírra!



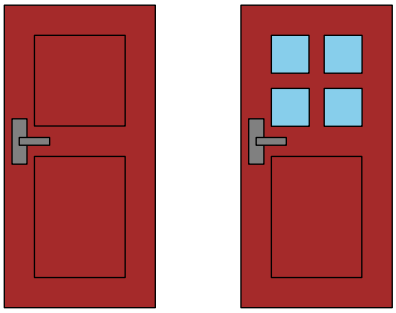
2) Írj programokat az alábbi házikók megrajzolására!



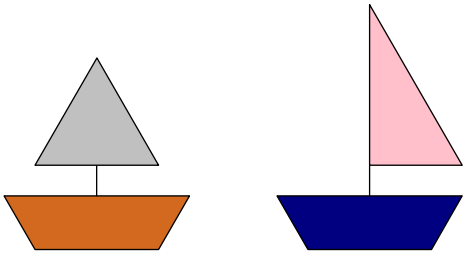
3) Készítsd el az alábbi ablakokat megrajzoló programokat!



4) Készíts el az alábbi két ajtó megrajzolásához szükséges utasításokat!



5) Készítsd el a következő két hajót megrajzoló programokat!



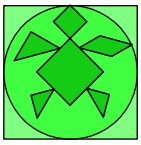
Extra parancsok

Az első fejezetben megtanultuk az alapvető teknőcmozgató utasításokat, tudunk színes vonalú és kitöltésű alakzatokat készíteni. Ezek birtokában már szinte bármilyen rajzot elkészíthetünk a teknőc segítségével. Ebben a fejezetben néhány extra utasítást fogunk elsajátítani, amelyek bővítik a rajzolási eszköztárunkat.

Kész eljárások

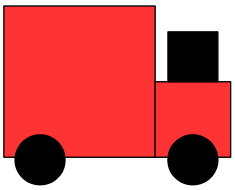
Az eddigiek során a teknőc csak egyenes vonalak megrajzolására volt képes. A most következő utasításokkal azonban kész alakzatokat készítettünk a teknőccel. Ezek az alakzatok pedig a *négyzet*, a *kör*, a *téglalap* és az *ellipszis*. Szerencsére a rajzoló utasítások neve megegyezik a megfelelő alakzat nevével. Így már ki is találhattad, hogy a

```
négyzet 50
kör 50
```



utasítások egy négyzetet és egy kört fognak rajzolni, mégpedig a teknőc alapértelmezett áttetsző zöld töltőszínével.[4] A négyzet utasítás paramétere a négyzet oldalának hosszát, a kör utasításé pedig a kör átmérőjének nagyságát határozza meg. Mindkét alakzatot úgy készíti el a teknőc, hogy a rajzolás előtt és után is a négyzet, illetve kör középpontjában helyezkedik el, továbbá a négyzet a teknőc irányának megfelelően van elforgatva a középpontja körül.

Gyakorlásképp készítsük el a következő mozdonyt megrajzoló utasításokat, a kör és a négyzet parancsok felhasználásával:



Segítségül elárulom, hogy a legnagyobb négyzet oldalának hossza kétszerese, illetve háromszorosa a kis négyzetek oldalhosszainak. A legkisebb négyzet oldal-

[4] A kör amiatt tűnik sötétebbnek, mint a négyzet, mert az áttetszőség miatt a színe összeadódik a négyzetével. Az egérrel odébb helyezve az alakzatokat, könnyedén kideríthető, hogy ugyanolyan színűek.

hossza megegyezik a kerék átmérőjével. Valamint a töltőszíneket mindig az alakzatok rajzolása előtt kell a megfelelő színre változtatni, hiszen a négyzet és a kör utasítás már eleve kitöltött alakzatokat készít.

Íme egy lehetséges megoldás:

```
töltőszín „piros”
négyzet 60
tollatfel hátra 15 jobbra 90 előre 45 balra 90 tollatle
négyzet 30
tollatfel előre 25 tollatle
töltőszín „fekete”
négyzet 20
tollatfel hátra 40 tollatle
kör 20
tollatfel balra 90 előre 60 tollatle
kör 20
```

A kör és a négyzet utasítás paraméterei az előre, hátra parancsoknál már megbeszélt mértékegységekben (pont, cm, mm) adhatóak meg.

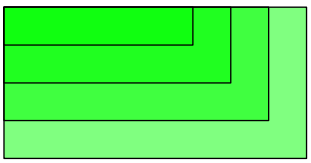
A téglalap és az ellipszis megrajzolásához már nem elegendő egyetlen paraméter. Vegyük először a téglalapot. A téglalapnak két különböző méretű oldala van, természetesen mindkét oldal hosszát meg kell adnunk, hogy a teknőc tudja, milyen téglalap rajzolását várjuk tőle. A két paramétert a következőképpen adjuk meg:

```
téglalap [60, 40]
```

Ez az utasítás egy olyan téglalapot rajzol, amelynek szélessége 60, a magassága pedig 40 pont. A téglalap oldalhosszait itt egy két elemű listával adtuk meg. A listákról részletesen a 10. fejezetben lesz szó. Most annyit kell tudni róluk, hogy a lista a szögletes zárójelek között vesszővel elválasztott elemek felsorolása.

Megjegyzés: Lista esetében nem lehet szóköz a felsorolt elemek és a szögletes zárójelek között!

Feladat: Készítsd el az alábbi ábrát megrajzoló programot! Használd a most tanult `téglalap` utasítást!

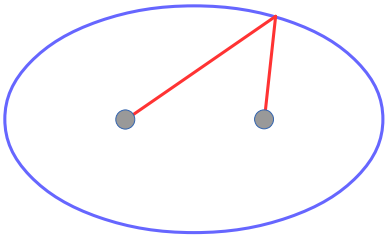


Egy lehetséges megoldás:

```
téglalap [40mm, 20mm]
tollatfel előre 2,5mm balra 90
előre 2,5mm jobbra 90 tollatle
téglalap [35mm, 15mm]
tollatfel előre 2,5mm balra 90
előre 2,5mm jobbra 90 tollatle
téglalap [30mm, 10mm]
tollatfel előre 2,5mm balra 90
előre 2,5mm jobbra 90 tollatle
téglalap [25mm, 5mm]
```

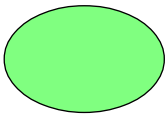
Mielőtt belekezdenénk az ellipszisrajzolásba, megbeszéljük, mi is az az ellipszis. Először nézzük meg, hogyan tudsz papíron ellipszist rajzolni. Ehhez kövesd a következő bekezdés utasításait!

Vegyél egy parafa táblát, erre helyezd rá a rajzlapod. Szúrj két rajzszöget a papírlapba egymástól néhány ujjnyi távolságra, majd köss rájuk egy madzagot, amely hosszabb, mint a két leszúrt rajzszög távolsága, de kisebb, mint annak a kétszerese. Akaszd a ceruzádat a madzagba, és óvatosan, a madzagot mindig feszesen tartva rajzolj a papírra. Egy, az alábbi ábrához hasonló alakzatot fogsz kapni.



Ez az ellipszis (az ábra a két rajzszöget és az ellipszis egy pontjába kifeszített, piros színnel rajzolt madzagot is mutatja). Az ellipszis pontjai: minden olyan pont, amely két rajzszögtől mért távolságának összege (amely megegyezik a madzag hosszával) állandó. Az ellipszis olyan, mint egy lapított kör. A kör méretét az átmérőjével adtuk meg, de az ellipszisnek nincs átmérője. Azonban a téglalaphoz hasonlóan az ellipszis is megadható két érték segítségével. Ez a két érték az ellipszis úgynevezett nagy- és kistengelyének a hosszai lesznek. Ezt úgy is elképzelheted, mint az ellipszis köré írt téglalap szélessége és magassága.

```
ellipszis [60, 40]
```

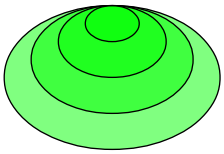


A teknőc az ellipszist és a téglalapot is úgy készíti el, hogy ő az alakzatok középpontjában helyezkedik el, továbbá mindkét alakzat a teknőc irányának megfelelően van elforgatva a középpontja körül.

Rajzolhatunk-e a téglalap utasítással négyzetet, illetve az ellipszis utasítással kört? A válasz természetesen az, hogy igen. Az alábbi két sor egy 40 pont oldalú négyzetet és egy 40 pont átmérőjű kört fog rajzolni.

```
téglalap [40, 40]
ellipszis [40, 40]
```

Feladat: Készítsd el az alábbi ábrát megrajzoló programot! Használd a most tanult ellipszis utasítást!



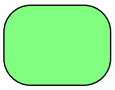
Egy lehetséges megoldás:

```
ellipszis [120, 80]
tollatfel előre 10 tollatle
ellipszis [90, 60]
tollatfel előre 10 tollatle
ellipszis [60, 40]
tollatfel előre 10 tollatle
ellipszis [30, 20]
tollatfel előre 10 tollatle
```

A téglalap és ellipszis utasításaink paraméterlistájának bővítésével további alakzatokat rajzolhatunk.

Készíthetünk például lekerekített sarkú téglalapot:

```
téglalap [40, 30, 10]
```



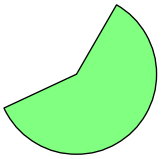
Ekkor egy harmadik paramétert is megadunk a téglalap utasításnak. Ez a paraméter a lekerekítés körívének sugarát állítja be.

Az ellipszis esetében még további három paraméterrel bővíthetjük az utasításunkat. Így készíthetünk például egy negyed ellipszist:

```
ellipszis [60, 40, 0, 90]
```

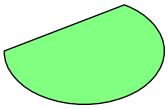
Itt a 0 és a 90 paraméter azt jelenti, hogy a 0 foktól a 90 fokig rajzoljuk meg az ellipszist. A fokok a teknőc aktuális irányától (ami a 0 foknak felel meg) az óramutató járásának irányába felvett szögeknek felelnek meg. Így például a következő utasítással készíthetünk egy 215 fokos (a teknőc irányához képest 30 és 245 fokban álló sugarú) körcikket:

```
ellipszis [60, 60, 30, 245]
```



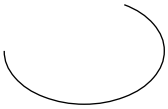
Arra is van mód, hogy ne ellipsziscikket, hanem ellipszisszeletet készítsünk, ha ötödik paraméternek a 2-es számot adjuk meg:

```
ellipszis [60, 40, 30, 270, 2]
```



Vagy csupán egy ellipszisívet, ha az ötödik paraméter a 3-as szám:

```
ellipszis [60, 40, 30, 270, 3]
```



Alakzatok rögzített elhelyezése

Rajzainkat eddig oda készítettük el, ahol a teknőc volt. Ha máshova szerettük volna a rajzunkat elhelyezni, akkor a teknőccel felemelt tollal oda kellett mennünk a megfelelő helyre. Most tanulunk két olyan parancsot, amelyek segítségével a lap bármelyik pontjába eljuthatunk.

Ahhoz, hogy megértsük a következő parancsok működését, néhány háttérismeretre is szükségünk lesz. Matematikaórán valószínűleg már találkoztál a koordináta-rendszerrel. Nos a teknőcünk is egy ilyen koordináta-rendszerben helyezkedik el, és a teknőc aktuális helyét pedig két koordinátával, a vízszintes és függőleges tengelyen mért értékekkel lehet leírni. A következő utasítással kiíratható a teknőc aktuális pozícióját:

```
ki hely
```

A teknőc alapértelmezett pozíciójához, az oldal középpontjához tartozó koordináták: [297,6377952755905, 420,859842519685].[5] A két értéket most is pontokban kaptuk meg.

Ha a hely parancsnak paraméterek is adunk meg, akkor a teknőcöt a megadott koordinátájú helyre küldhetjük. Mivel a koordinátákat két számmal tudjuk megadni, most is listát használunk:

```
hely [200, 200]
```

Így a (200, 200) koordinátájú pontokba jutottunk. Közben megfigyelhetted, hogy a teknőc eközben bizony nyomot hagyott a papíron (ha ezt nem szeretnénk, akkor fel kell emelnünk a tollát). Valamint már nem felfelé néz, hanem a haladásának megfelelő irányba. Mennyit forduljon, hogy ismét felfelé nézzen? Bizony ezt már nagyon nehéz megmondani. Ezért lesz szükség az irány parancsra. Először most is kiíratjuk a teknőc aktuális irányát:

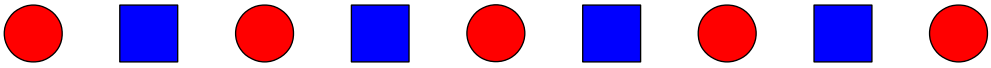
```
ki irány
```

Megtudtuk, hogy a teknőcünk 336,15°-os irányba néz. Most még azt kell kiderítenünk, hogy honnan mérjük a szöget, és hol van a koordináta-rendszer origója? Ehhez írjuk be a következő két sort:

```
hely [0, 0]
irány 0
```

Az origó tehát a lap bal felső sarka, a 0 irány pedig a felfelé nézésnek felel meg.

Feladat: Készíts színes négyzetekből és körökből álló sormintát. A sorminta a lap széleitől 2 cm-re kezdődjön, az alakzatok szélessége 1-1 cm legyen, és egymástól azonos távolságra helyezkedjenek el.



Egy lehetséges megoldás:

```
tollatfel hely [25mm, 25mm] irány 0 tollatle
töltőszín „piros” kör 1cm
tollatfel hely [45mm, 25mm] irány 0 tollatle
töltőszín „kék” négyzet 1cm
tollatfel hely [65mm, 25mm] irány 0 tollatle
töltőszín „piros” kör 1cm
tollatfel hely [85mm, 25mm] irány 0 tollatle
töltőszín „kék” négyzet 1cm
tollatfel hely [105mm, 25mm] irány 0 tollatle
```

[5] Az alapértelmezett A4-es oldal esetén.

```
töltőszín „piros” kör 1cm
tollatfel hely [125mm, 25mm] irány 0 tollatle
töltőszín „kék” négyzet 1cm
tollatfel hely [145mm, 25mm] irány 0 tollatle
töltőszín „piros” kör 1cm
tollatfel hely [165mm, 25mm] irány 0 tollatle
töltőszín „kék” négyzet 1cm
tollatfel hely [185mm, 25mm] irány 0 tollatle
töltőszín „piros” kör 1cm
```

További tollbeállítások

A teknőc tollának nem csupán a színét és a fel-, illetve lerakott állapotát tudjuk beállítani, hanem például a toll vastagságát és az általa húzott vonal stílusát is. Mivel ez is tollállapot, csak akkor látjuk a megfelelő utasítás eredményét, ha rajzolunk utána valamit. Próbáld ki a következő sorokat:

```
jobbra 90
tollvastagság 3
előre 100
```


A tollvastagságot, ahogy a tollszínt is leolvashatjuk a teknőc körvonaláról. Ha ismét az alapértelmezett félpontnyi vonalvastagsággal szeretnénk rajzolni, akkor a tollvastagság 0,5 parancsot kell kiadnunk. Rövid változat: tv 0,5.

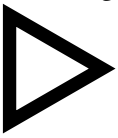
Megjegyzés: a tollvastagság 0 *parancs az elérhető legvékonyabb vonalat jelenti (nyomtatásbeli vastagságát a nyomtató képességei határozzák meg).*

Most rajzoljunk egy vastag körvonalú háromszöget!

```
tollvastagság 5
előre 40 jobbra 120
előre 40 jobbra 120
előre 40 jobbra 120
```

Ha nagyobb tollvastagságot használunk, akkor sarkoknál megfigyelhetjük, hogy az lekerekített és nem hegyes. Ha mégis hegyes sarkokat szeretnénk a háromszögünknek, akkor a fenti kódot ki kell egészítenünk a következőképpen:

```
tollvastagság 5
tollsarok „hegyes”
előre 40 jobbra 120
előre 40 jobbra 120
előre 40 jobbra 120
```



Kipróbálhatjuk a tollsarok utasítást a „tompa” paraméterrel is, amellyel lecsapott végű sarkokat kapunk. Amennyiben ismét kerek sarkokkal szeretnénk tovább rajzolni, adjuk ki a tollsarok „kerek” utasítást.

Lehetőség van a vonalvégek vonalvastagságnak megfelelő megnyújtására is, amely lehet az alapértelmezetthez hasonlóan négyzetes, valamint lekerekített is:

```
tollvastagság 15
tollhegy „négyzet” jobbra 90
előre 40 tollatfel előre 20 tollatle
tollhegy „kerek”
előre 40 tollatfel előre 20 tollatle
tollhegy „nincs”
előre 40
```

A teknőcünk nemcsak vastag, illetve vékony vonallal, hanem háromféle tollstílussal is képes rajzolni: folyamatos, pontozott, szaggatott. Mielőtt kipróbálod a következő kódot, töröld le a képernyőt, és küldd a teknőcöt haza!

```
jobbra 90 tollvastagság 2
tollstílus „pontozott”
előre 50
tollstílus „szaggatott”
előre 50
tollstílus „folyamatos”
előre 50
```


Az említett tollállapotokat természetesen tetszőlegesen kombinálhatjuk.

Feladat: Rajzolj egy narancssárga színű, szaggatott, 3 pont vastagságú, 3 cm hosszú, vízszintes vonalat!

Megoldás:

```
jobbra 90
tollszín „narancs”
tollvastagság 5
tollstílus „szaggatott”
előre 3cm
```

Már biztosan eszedbe jutott, hogy ha ennyiféle tolla van a teknőcnek, akkor biztosan az alakzatok kitöltésénél is van még valami a tarsolyában. Ez így is van! A töltőstílus utasítás paraméterezése azonban elég összetett, egy négyelemű listával tudjuk megadni a paramétereit. Például:

```
töltőstílus [1, „piros”, 5pt, 45°]
```

```
négyzet 1cm
```

Ez az utasítás a képen látható kitöltési mintázatot hozza létre, a félig áttetsző zöld szín a teknőc alapértelmezett kitöltőszíne, ezen látunk egy vonalkázást (1. paraméter), amely piros színű (2. paraméter), a vonalak távolsága 2 pont (3. paraméter) és 45°-os szögben állnak (4. paraméter).

A töltőstílus első paramétere 4-féle lehet. A 0 töltőstílus az alapértelmezett, ekkor csak a kitöltőszín látható, az 1-es az imént bemutatott vonalkázás, a 2-es pedig a négyzetrács:

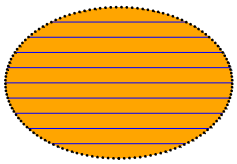
```
töltőstílus [2, „piros”, 5pt, 45°]
négyzet 1cm
```

A 3. az 1 és 2 vonalkázás ötvözete, azaz egy négyzetrács és azzal 45° bezáró vonalkázás:

```
töltőstílus [3, „piros”, 5pt, 45°]
négyzet 1cm
```

Az utóbbi három töltőstílusnál beállíthatjuk a vonalak színét, távolságát és a vízszintestől mért szögét, ahogy a mintákban is láthattuk.

Feladat: Rajzolj egy 3 cm széles, 2 cm magas ellipszist, amelynek kitöltőszíne narancssárga, és vízszintesen vonalkázott kék csíkokkal, amelyek 2 mm távolságra vannak egymástól. Az ellipszis körvonala legyen pontozott!

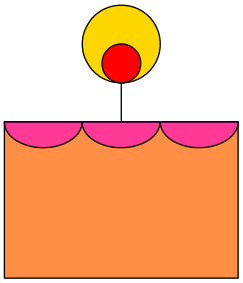


Megoldás:

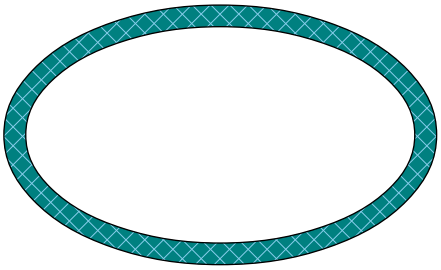
```
töltőszín „narancs”
töltőstílus [1, „kék”, 2mm, 0]
tollstílus „pontozott”
ellipszis [3cm, 2cm]
```

Feladatok

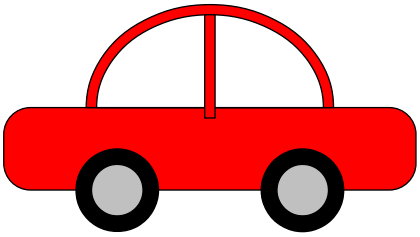
6) Készíts az alábbi minta alapján tortát a most tanult utasítások segítségével!



7) Készítsd el az alábbi képkeret programját!

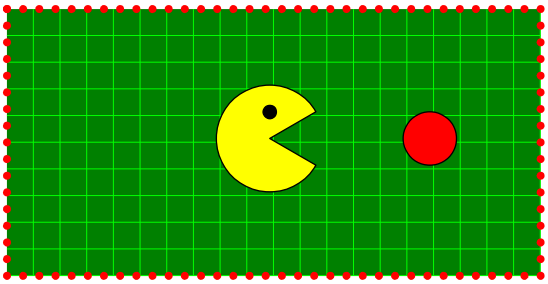


8) Készítsd el a képen látható kisautót megrajzoló programot, használd a téglalap és az ellipszis utasításokat!



9) Készíts Android emberkét rajzoló programot (az alábbi mintában használt zöld szín RGB kódja: [153, 204, 102])

10) Készítsd el az alábbi Pacman figurás ábrát a tanult utasítások segítségével!

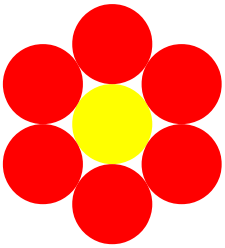


Alakzatok rajzolása ismétléssel

Az előző két fejezetben megismerkedtünk az alapvető teknőcmozgató utasításokkal. Megtanultuk, hogyan lehet a teknőc tollának tulajdonságait módosítani (pl. tollszín, tollvastagság) valamint elsajátítottuk a téglalap-, kör- és ellipszisrajzolás fortélyait.

Ezekkel az ismeretekkel most már tetszőleges alakzatokat képesek vagyunk megrajzolni. Ebben a fejezetben azt fogjuk megtanulni, hogyan tudjuk a rajzainkban felbukkanó ismétlődéseket kihasználni ahhoz, hogy rövidebb és átláthatóbb utasításokkal készítsük el az ábráinkat.

Vegyük példának az alábbi képen látható virágot:



Ennek megrajzolása a következő utasításokkal történhet:

```
tollszín „láthatatlan” töltőszín „sárga”
kör 30
töltőszín „piros” előre 30 jobbra 60
kör 30 jobbra 60 előre 30
kör 30 jobbra 60 előre 30
kör 30 jobbra 60 előre 30
kör 30 jobbra 60 előre 30
kör 30 jobbra 60 előre 30
kör 30 jobbra 60 előre 30
balra 60 hátra 30
```

Ha megnézed a kódot, akkor láthatod, hogy a 4. sor hatszor ismétlődik egymás után, ezek az utasítások rajzolják meg a virág szirmait. Milyen jó lenne, ha ezt az ismétlődést rövidebben is megadhatnánk, ugye? Szerencsére erre van lehetőségünk, mégpedig az ismétlés parancs segítségével. Ez az utasítás egy kicsit más, mint a korábbiak, eddig voltak paraméter nélküli (pl. tollatfel, tollatle, törölkép, haza stb.) és paraméteres (előre, hátra, jobbra, balra, tollszín, töltőszín, téglalap, kör, ellipszis stb.) utasításaink.

Az ismétlés parancsnak két paramétere van: az első paraméterrel adjuk meg az ismétlés darabszámát, a második paraméterrel pedig azoknak az utasításoknak a listáját, amelyeket többször végre kell hajtani. Ahhoz, hogy tudjuk, mely parancsokat kell ismételni, azokat [ ] közé tesszük. Tehát az ismétlés parancs szerkezete a következő:

ismétlés *szám* [*utasítások listája*]

Fontos, hogy a nyitó szögletes zárójel ([) után és a záró szögletes zárójel (]) elé tegyünk egy-egy szóközt. Ennek okáról a listákról szóló fejezetben lesz majd szó.

Rövid változat:

ism *szám* [*utasítások listája*]

A fentiek alapján már kitalálhatod, hogy a virágot a következő kóddal is megrajzolhatjuk:

```
tollszín „láthatatlan” töltőszín „sárga”
kör 30
töltőszín „piros” előre 30 jobbra 60
ismétlés 6 [ kör 30 jobbra 60 előre 30 ]
balra 60 hátra 30
```

Ha kipróbáltad a fenti példakódot, láthattad, hogy a] jel utáni utasítások már csak egyszer hajtódnak végre, miután az ismétlések befejeződtek.

Most nézzünk néhány példakódot! Vajon mit rajzolnak az alábbi utasítások?

```
ismétlés 3 [ előre 50 jobbra 120 ]
ismétlés 5 [ előre 40 hátra 40 jobbra 72 ]
ismétlés 6 [ jobbra 60 előre 30 ]
```

Biztosan magad is kitaláltad, hogy az első példakód egy 50 pont oldalú szabályos háromszöget rajzol. Ennek az utasítássorozatnak a segítségével megrajzolhatjuk a következő fenyőfát:

Ha ránézünk a fenyőfára, rögtön szembe tűnik, hogy a lombja három kitöltött háromszögből áll, amelyeket egymás felett kicsit elcsúsztattunk. Ez alapján ezt a fenyőfát például a következő utasítássorozattal tudjuk elkészíteni:

```
tollszín „barna” tollvastagság 5
előre 20 tollvastagság 0
tollatfel jobbra 90 hátra 20 balra 90 tollatle
töltőszín „zöld” tollszín „zöld”
jobbra 30
ismétlés 3 [ előre 40 jobbra 120 ] tölt
balra 30
tollatfel előre 20 tollatle
jobbra 30
ismétlés 3 [ előre 40 jobbra 120 ] tölt
balra 30
tollatfel előre 20 tollatle
jobbra 30
ismétlés 3 [ előre 40 jobbra 120 ] tölt
balra 30
tollatfel előre 20 tollatle
```

A kódban rögtön felfedezzük az ismétlődést, tehát az ismétlés parancsot ismét bevethetjük a kód lerövidítésére az alábbi módon:

```
tollszín „barna” tollvastagság 5
előre 20 tollvastagság 0
tollatfel jobbra 90 hátra 20 balra 90 tollatle
töltőszín „zöld” tollszín „zöld”
ismétlés 3 [
    jobbra 30
    ismétlés 3 [ előre 40 jobbra 120 ] tölt
    balra 30
    tollatfel előre 20 tollatle
]
```

Ebben a példakódban megfigyelhetted, hogy az ismétlés parancs utasításlistája tartalmazhat további ismétlés utasítást is. Továbbá az is feltűnhet, hogy az eddig megszokottól kicsit eltérően tagoltuk a kódot. A program szempontjából ez a tagolás nem lényeges, azonban a továbbiakban érdemes lesz ezt a fajta kódtagolást követned, amennyiben az ismétlés utasításlistája több sorból áll. A példaprogramokban én is ezt teszem majd. Hogy miért hasznos ez? Azért, mert így könnyen áttekinthető lesz a kódunk, első pillantásra látjuk, hogy mely utasítások kerülnek ismétlésre. Az ismétlendő utasításokat tehát ezután majd mindig beljebb kezdjük, amit úgy tudunk megtenni, hogy a sor elején lenyomjuk a *Tab* billentyűt (általában a billentyűzet bal szélén, a *CapsLock* billentyű felett található).

Most már majdnem mindent tudunk, amit az ismétlés parancsról tudni kell. Mindjárt jönnek is a gyakorló feladatok. De előtte próbáld ki, mi történik akkor, ha nem adod meg az ismétlésszámot! Például:

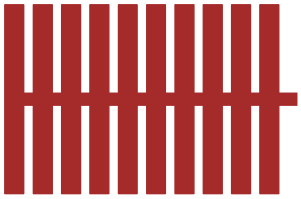
```
ismétlés [ előre 100 jobbra 90 ]
```

A teknősünk a fenti utasítások hatására körbe-körbe szaladgál egy négyzet körvonalán. Abba sem hagyja, amíg a *Leállítás* gombra nem kattintunk. Ilyenkor végtelen ismétlődést idézünk elő, ezt a programozók végtelen ciklusnak nevezik. Felmerülhet benned a kérdés, hogy miért nem kapunk ehelyett inkább valamilyen hibaüzenet, hogy elrontottuk az ismétlés utasítást. Ennek természetesen van oka, erről később a 6. fejezetben lesz majd szó.

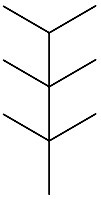
Feladatok

11) Készíts programot a szabályos ötszög megrajzolására! Mit kell módosítani a programon, hogy ne ötszöget, hanem szabályos 6-, 7-, 8-, 9- stb. szöget rajzolhassunk vele?

12) Rajzoltasd meg a teknőccel a mintán látható kerítést!



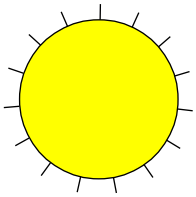
13) Készítsd el az alábbi ágat megrajzoló programot! Az ágat alkotó szakaszok azonos hosszúságúak.



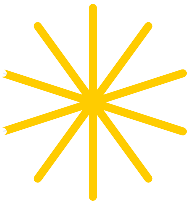
14) Az előző példában készített program felhasználásával készítsd el a következő hópelyhet! Megtaláltad az ismétlődő alakzatot?

15) Tervezz saját hópelyhet és készíts hozzá Logo programot!

16) Készíts programot, amely a következőhöz hasonló napocskát rajzol!



17) Rajzoltass csillagot a teknőccel minél többféleképpen! Segítségül adunk két példát, de találhatsz ki másféle csillagot is!

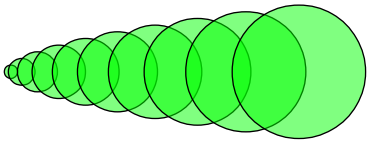


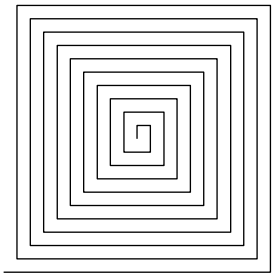
18) Készíts a teknőc segítségével hóembert!

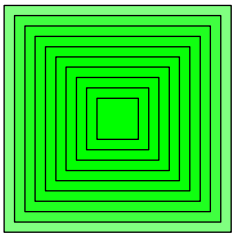
Változók használata alakzatok rajzolásához, matematikai műveletek

Ennek a fejezetnek a címe egy kicsit csalóka. Ugyanis a hagyományos értelemben vett változókat majd csak az adatbeolvasással foglalkozó fejezetben fogom bemutatni. Most csak ízelítőt adok a változókról, de a könyv további részében is sok helyen előfordulnak majd.

Az első változó, amellyel foglalkozni fogunk, a LibreLogo egy beépített változója, amelyet ismétléses programszerkezeteknél tudunk használni. Ennek a bemutatásához gondolkodjunk egy kicsit azon, mire lenne szükségünk az eddigi ismereteink mellé a következő ábrák ismétléssel való megrajzolásához!







Az első ábra esetében 10 darab kört rajzolunk, egyre nagyobb sugárral és egyre nagyobb távolságra egymástól. Az első kör átmérője 5, a másodiké 10, a harmadiké 15 és a 10-iké pedig 50 pont. Tehát tudnunk kellene, hogy az ismétlés hányadik lépésénél tartunk. Valóban, a hányadik nevű változó értéke veszi fel az ismétlések között az ismétlés sorszámának értékét, először 1-et, majd 2-t, 3-at, 4-et stb. és végül a 10-et. A hányadik változó használata a következőképpen történhet az első ábra megrajzolásában:

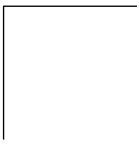
```
jobbra 90
ismétlés 10 [
    kör hányadik * 5
    tollatfel előre hányadik * 2 tollatle
]
```

A mintában láthatod, hogy a körök közötti távolság arányos növekedését is a hányadik változó segítségével értük el. A * jel a szorzást jelenti a fenti kódban. Ennek kapcsán említést kell tennünk arról, hogy természetesen a teknőcünk számolni is képes. Ismeri az alábbi táblázatban összefoglalt műveleteket:

+	összeadás	//	maradékos osztás hányadosa
–	kivonás	%	osztási maradék
*	szorzás	**	hatványozás
/	osztás		

A műveletek sorrendje megegyezik a matematikaórán tanultakkal, valamint ugyanazon szabályok szerint zárójelezhetjük a műveleteket. A példából az is látszik, hogy az utasítások paraméterének is adhatunk meg ilyen kiszámítandó értékeket.

A második ábra megrajzolásához egy-egy fél négyzetet rajzoltunk az ismétlés minden lépésében, egyre nagyobb oldalhosszakkal.

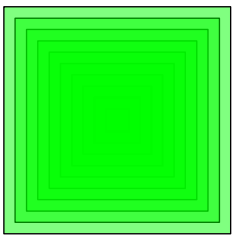


```
ismétlés 20 [
    előre hányadik*5 jobbra 90
    előre hányadik*5 jobbra 90
]
```

A harmadik ábra megrajzolásához méltán jut eszedbe a következő megvalósítás:

```
ismétlés 10 [
    négyzet hányadik*5
]
```

Azonban így a következő ábrához jutunk:



Rossz a négyzeteink sorrendje! Hiszen a legkisebb négyzet megrajzolásával kezdünk és utána fölé rajzoljuk az egyre nagyobb négyzeteket. Nyilvánvaló, hogy az ismétlést nem játszhatjuk le visszafelé. Mit tehetünk hát? Természetesen a rajzolást a legnagyobb négyzettel kell kezdeni és utána csökkenteni minden lépésben a négyzetek oldalainak hosszát. Erre mutat egy módszert a következő példa:

```
ismétlés 10 [
    négyzet 55-hányadik*5
]
```

Így az első négyzet mérete 55–1·5=50, a másodiké 55–2·5=45, a harmadiké 55–3·5=40 stb. pont lesz.

A hányadik tehát egy változó, amely az ismétlésen belül kap értéket attól függően, hogy az ismétlés hányadik lépésénél tart a program. Az ilyen változókat hívjuk a programozásban ciklusváltozónak. A ciklusváltozó az ismétlésen (ciklusson) kívül nem létezik. Ha a hányadik változót az ismétlés előtt vagy után szeretnénk használni, akkor hibaüzenetet kapunk!

A programozásban nem csak ciklusváltozók léteznek, hanem egyéb változók is. Ezek valamiben hasonlítanak, néhány dologban pedig különböznek a hányadik változótól. A változók egy dobozként foghatóak fel, amelyekben a feladatmegoldás közben eltárolhatunk különböző értékeket. Az ismétlés során a hányadik változóban tárolódik el az ismétlési lépés sorszáma.

Minden változónak van neve (pl. hányadik), és van értéke. Azért hívjuk változónak, mert az értéke a program futása közben változhat. Mint a hányadik változó érteke, amely az ismétlés minden lépésében más és más. A változó értékére a változó nevével hivatkozunk, ahogyan ezt a hányadik esetében meg is tettük. A hányadik változó azonban az ismétléskor automatikusan rendelkezésünkre állt. Hogyan használhatunk további változókat? A következőképp:

```
méret = 100
```

Tehát választunk egy változónevet (méret), valamint ennek a változónak adunk egy értéket (100). Az értékadás az = jellel történik. Az értékadáskor fontos, hogy a változó neve szerepeljen az egyenlőségjel bal oldalán.

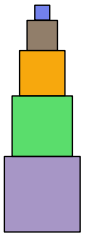
Milyen változóneveket választhatunk?

Nem használhatjuk változónévként a LibreLogo beépített parancsait és azok rövidítéseit, valamint a változónevek nem kezdődhetnek számmal, nem tartalmazhatnak szóközt, zárójeleket és műveleti jeleket.

A változók értéke nagyon sokféle lehet: szám, szöveg, karakter (betű) valamint összetett értékek is, mint a későbbiekben tárgyalt lista, halmaz vagy szótár. A változókat sokféle probléma megoldásában használjuk, sőt a használatuk a legtöbb feladatban nélkülözhetetlen.

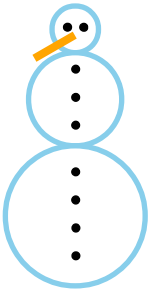
Feladatok

19) Készítsd el az alább építőkockákból álló tornyot megrajzoló programot!



20) Készíts programot, amely az alábbi nyakláncot rajzolja meg!

21) Készíts programot, amellyel az alábbi hóembert lehet megrajzolni!



Eljárások és paraméterek, rajzolt alakzatok csoportba foglalása

Ebben a fejezetben azt fogjuk megtanulni, hogyan tudunk a teknőcnek új parancsszavakat tanítani, amely az összetett rajzok elkészítésében nagy segítségünkre lesz. Ezzel párhuzamosan megismerkedünk a programozás egy fontos lépésével, mégpedig a tervezéssel is. A fejezet végén pedig szó lesz egy hasznos parancsról, amelynek segítségével a több alakzatból álló rajzainkat egy képpé forraszthatjuk össze.

Új parancsszavak megtanítása

Emlékezzünk vissza a fenyőfát megrajzoló programunkra:

```
tollszín „barna” tollvastagság 5
előre 20 tollvastagság 0
tollatfel jobbra 90 hátra 20 balra 90 tollatle
töltőszín „zöld” tollszín „zöld”
ismétlés 3 [
    jobbra 30
    ismétlés 3 [ előre 40 jobbra 120 ] tölt
    balra 30
    tollatfel előre 20 tollatle
]
```

A kiemelt kódrészlet egy kitöltött háromszöget rajzol. Ha lenne háromszög utasításunk, akkor a következőképp egyszerűsödne a fenti kód:

```
tollszín „barna” tollvastagság 5
előre 20 tollvastagság 0
tollatfel jobbra 90 hátra 20 balra 90 tollatle
töltőszín „zöld” tollszín „zöld”
ismétlés 3 [
    jobbra 30
    háromszög
    balra 30
    tollatfel előre 20 tollatle
]
```

Ahhoz, hogy valóban használhassuk ezt az utasítást, meg kell tanítanunk a teknőcnek a háromszög parancsot, és hogy mit tegyen a parancs hatására. Ezt a következőképpen tehetjük meg:

```
eljárás háromszög
    ismétlés 3 [ előre 40 jobbra 120 ] tölt
vége
```

Ha a fenti háromsoros kódot beírjuk és elindítjuk a programot, akkor azt tapasztaljuk, hogy a teknőc egy helyben marad, nem csinál semmit. Ez azért van így, mert a fenti utasítással csak megtanítottuk a teknőcnek a háromszög rajzoló utasítást, de még nem kértük meg rá, hogy rajzoljon nekünk egy háromszöget, hiszen még nem adtuk ki a háromszög parancsot. Tegyük meg:

```
háromszög
```

Fontos, hogy a háromszög utasítást csak azután adjuk ki, hogy megtanítottuk a teknőcnek. Az alábbi utasítás tehát hibát fog jelezni.

Helytelen kód:

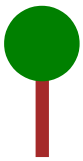
```
háromszög
eljárás háromszög
    ismétlés 3 [ előre 40 jobbra 120 ] tölt
vége
```

Az eljárás megadása után viszont bármikor adjuk ki a háromszög parancsot, akkor a teknőc háromszöget rajzol az aktuális helyzetének, irányának, toll- és töltőszínének megfelelően.

Az új utasítások (eljárások) nevének megadásakor a változóneveknél már említett feltételeket kell betartanunk: nem használhatjuk eljárásnévként a már meglévő Logo parancsokat és azok rövidítéseit, nem kezdődhetnek számmal, nem tartalmazhatnak műveleti jeleket és zárójeleket.

Feladat: Az alábbi eljárás egy fát rajzol. Hol van a teknőc a fa kirajzolása előtt és után?

```
eljárás fa
    tollszín „barna” tollvastagság 5 előre 60 tollvastagság 0
    töltőszín „zöld” tollszín „láthatatlan” kör 40
vége
fa
```



A teknőc a fa kirajzolása előtt a törzs aljánál található, míg az eljárás végén a lombkorona közepén. Valamint a teknőc tollának színe láthatatlan lesz a fa utasítás végrehajtása után. A korábban megtanult négyzet, kör, téglalap és ellipszis eljárásainknál már megszoktuk, hogy a teknőc rajzolás előtti és utáni állapotai megegyeznek. Ez megkönnyítette az utasítások használatát bonyolultabb ábrák készítése során. Ezért a továbbiakban az alábbi szabályhoz tartjuk magunkat:

Az eljárásaink készítésekor törekszünk arra, hogy a teknőc rajzolás előtti és utáni állapotai megegyezzenek, azaz állapotátlátszóak legyenek.

Ettől a szabálytól csak abban az esetben térünk el, ha az a feladat megoldását kifejezetten megkönnyíti.

Módosítjuk a fa eljárást a következőképpen:

```
eljárás fa
    tollszín „barna” tollvastagság 5 előre 60
     tollvastagság 0
    töltőszín „zöld” tollszín „láthatatlan” kör 40
     tollatfel hátra 60 tollatle
vége
fa
```

Így a teknőc ugyanott van a rajzolás előtt és után. Mi a helyzet azonban a tollának színével, vastagságával, valamint a töltőszínével? Ezek bizony változhattak. Hogy a teknőc ezen állapotait is visszaállítsuk szükségünk lesz három változóra, amelyekben eltároljuk a teknőc tollának színét, a tollvastagságot és a töltőszínt, ehhez tudnunk kell, hogy a tollszín, tollvastagság, töltőszín utasításokat paraméter nélkül kiadva, a teknőc aktuális állapotait kapjuk vissza.

```
eljárás fa
     szín1 = tollszín
    szín2 = töltőszín
    vastagság = tollvastagság
    tollszín „barna” tollvastagság 5 előre 60
     tollvastagság 0
    töltőszín „zöld” tollszín „láthatatlan” kör 40
    tollatfel hátra 60 tollatle
     tollszín szín1
```

```
    töltőszín szín2
    tollvastagság vastagság
vége
fa
```

Tehát az eljárás elején eltároltuk a teknőc tollállapotait, majd a végén a változók segítségével visszaállítjuk az eltárolt állapotokat.

Paraméteres eljárások készítése

Az eddig általunk készített utasítások adott méretű alakzatokat rajzoltak. Ezek szerint, ha kisebb vagy nagyobb háromszöget, fát stb. szeretnénk rajzolni, akkor új eljárást kell készítenünk? Szerencsére nem! A saját eljárásainknak is adhatunk meg paramétereket, így az alakzatunk néhány tulajdonságát majd csak az utasítás kiadásakor adjuk meg, viszont azt már az eljárás megalkotásakor tudnunk kell, hogy melyek ezek a tulajdonságok.

Például szeretnénk egy olyan háromszög eljárást készíteni, amely adott oldalhosszúságú háromszöget rajzol. Akkor azt a következőképpen tehetjük meg:

```
eljárás háromszög oldal
    ismétlés 3 [ előre oldal jobbra 120 ]
vége
```

Az eljárás neve után megadtuk, hogy az eljárásnak van egy oldal nevű paramétere, az eljárás belsejében pedig azt, hogy ez az oldal paraméter a háromszög oldalának hosszát fogja jelenteni. Most már a háromszög utasításunk csak akkor rajzol háromszöget, ha megadjuk a háromszög oldalhosszát is, különben hibaüzenetet kapunk.

```
háromszög 40
```

Készítsük el a fa eljárás paraméteres változatát is, természetesen most is betartjuk az állapotmegtartási szabályt. A fa eljárásunk méret paramétere a fa magasságát jelenti.

```
eljárás fa méret
    szín1 = tollszín
    szín2 = töltőszín
    vastagság = tollvastagság
    tollszín „barna” tollvastagság méret/10 előre méret*3/4
     tollvastagság 0
    töltőszín „zöld” tollszín „láthatatlan” kör méret*1/2
    tollatfel hátra méret*3/4 tollatle
```

```
    tollszín szín1
    töltőszín szín2
    tollvastagság vastagság
vége
fa 80
```

A paraméteres eljárásnál még arra is ügyelnünk kell, hogy az alakzat minden részlete méretarányos legyen, ezért a méret paramétert használjuk a törzs magasságának és vastagságának, valamint a lombkorona méretének beállításához is.

Egy eljárásban használhatunk már meglévő eljárásokat, például az imént elkészült eljárásunk segítségével készíthetünk fenyőrajzoló eljárást is:

```
eljárás háromszög méret
    jobbra 30
    ismétlés 3 [ előre méret
        jobbra 120 ]
    tölt
    balra 30
vége
eljárás fenyő méret
    tollszín „barna” tollvastagság 5 előre méret/2 tollvastagság 0
    tollatfel jobbra 90 hátra méret/2 balra 90 tollatle
    töltőszín „zöld” tollszín „zöld”
    ismétlés 3 [
        háromszög méret
        tollatfel előre méret/2 tollatle ]
    tollatfel hátra 2*méret jobbra 90
    előre méret/2 balra 90 tollatle
vége
```

Csoportosítás

A LibreLogóval készített alakzatainkra kattintva zöld fogantyúk jelennek meg az alakzat körül (a sarkainál és az oldalainál). Ezeknél a fogantyúknál fogva a rajz átméretezhető. A fogantyúk jelzik, hogy az adott alakzat van kijelölve. Ekkor az alakzatot a kurzormozgató billentyűkkel (nyilakkal) vagy az egérrel elmozgathatjuk a képernyőn.

A rajzolás során mi magunk adjuk meg, hogy mely rajzrészek tartoznak egy alakzathoz. Alapesetben a tolltulajdonságok változtatása, az alakzat kitöltése,

négyzet, kör, téglalap, ellipszis rajzolása után kezdünk új alakzatot (toll felemelésével nem).

Ha mégis különböző színű, több négyzetet, kört, téglalapot, ellipszist tartalmazó alakzatot szeretnénk egy objektumként kezelni, akkor az őket megrajzoló utasításokat a kép utasítás után szögletes zárójelek közé kell elhelyezni. Például a fa eljárás által kirajzolt alakzatokat így forrasztjuk össze egy képpé:

```
kép [ fa 80 ]
```

Más néven ezt a műveletet az alakzatok csoportba foglalásának is hívjuk. A szögletes zárójelek és az utasítások közé most is szóközöket vagy pedig sortörést kell tennünk.

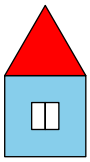
Persze nem csak egy eljárás eredményét foglalhatjuk csoportba, hanem több utasítást is, mint például a korábban megrajzolt kisvonat kódját:

```
kép [
    töltőszín „piros”
    négyzet 60
    tollatfel hátra 15 jobbra 90 előre 45 balra 90 tollatle
    négyzet 30
    tollatfel előre 25 tollatle
    töltőszín „fekete”
    négyzet 20
    tollatfel hátra 40 tollatle
    kör 20
    tollatfel balra 90 előre 60 tollatle
    kör 20
]
```

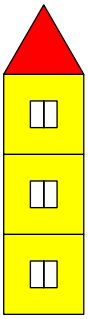
Feladatok

22) Készítsd az alábbi rajzokat előállító eljárásokat először paraméterek nélkül, majd alakítsd át őket paraméteressé! Minden esetben a kirajzoláskor foglald csoportba az alakzatokat!

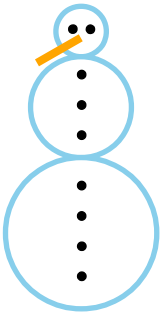
a) A ház eljárás paramétere az alapjának oldalhossza legyen!



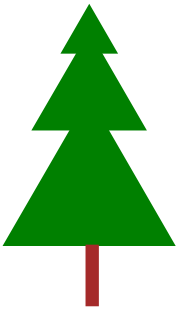
b) Az emeletes ház eljárás paramétere az alapjának oldalhossza és az emeletek száma legyen! (Segítség: Készíts emelet néven segédeljárást, amely egy emeletet rajzol meg.)



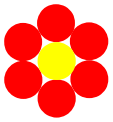
c) A hóember eljárás paramétere a fej átmérője legyen!



d) A fenyő eljárás paramétere a legkisebb háromszög oldalhossza legyen!



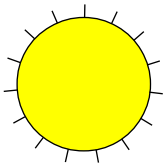
e) A virág eljárás paramétere a szirmok átmérőjének nagysága legyen!



f) A hópehely eljárás paramétere a legrövidebb szakaszok hossza legyen!

g) A csillag eljárás paramétere a csillag ágainak hossza és az ágak száma legyen!

h) A nap eljárás paramétere a napkorong átmérője legyen!



Elágazás, logikai műveletek

Az elágazás programszerkezettel való ismerkedést kezdjük egy feladattal. Készítsünk kerítést rajzoló eljárást, amelynek a paramétere a kerítés léceinek darabszáma.

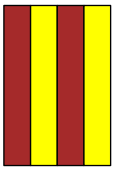
```
eljárás léc méret
    téglalap [méret/6, méret]
vége
eljárás kerítés lécdb
    szín1 = töltőszín töltőszín „barna”
    ismétlés lécdb [
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
    ]
     tollatfel jobbra 90 hátra lécdb*10 balra 90 tollatle
    töltőszín szín1
vége
kerítés 5
```

Hogyan módosítanánk a kerítés eljárást, ha azt szeretnénk, hogy a kerítés lécei felváltva barna és sárga színűek legyenek? Jó ötletnek tűnik, hogy a léceket párosával rajzoljuk ki, ekkor az ismétlések száma a lécek számának a felével lesz egyenlő.

```
eljárás kerítés lécdb
    szín1 = töltőszín
    ismétlés lécdb/2 [
         töltőszín „barna”
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
        töltőszín „sárga”
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
    ]
     tollatfel jobbra 90 hátra lécdb*10 balra 90 tollatle
    töltőszín szín1
vége
kerítés 6
```

Az eljárásunk helyesen működik, ha a lécek száma páros. Azonban, ha páratlan, akkor eggyel kevesebb lécet rajzolunk ki.

```
kerítés 5
```



Páros lécszám esetén tehát jó az eljárásunk, páratlan lécszám esetén azonban még egy barna lécet ki kell rajzolnunk. Tehát valamilyen feltételtől függően kell csak valamit a teknőcnek megrajzolni. Az ilyen feladatok megoldására használjuk az elágazásokat, amelyek a LibreLogo esetén a ha szerkezettel valósíthatók meg. Az alábbi példában megmutatjuk, hogyan:

```
eljárás kerítés lécdb
    szín1 = töltőszín
    ismétlés lécdb/2 [
        töltőszín „barna”
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
        töltőszín „sárga”
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
    ]
     ha lécdb % 2 == 1 [
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
    ]
    tollatfel jobbra 90 hátra lécdb*10 balra 90 tollatle
    töltőszín szín1
vége
```

A ha utasítást mindig egy feltétel követi, amely vagy igaz, vagy hamis. A feltételt követő szögletes zárójelekkel határolt utasítások csak akkor hajtódnak végre, ha a feltétel igaz volt, különben nem.

A fenti példában a % művelet a maradékképzést jelenti, azaz a (lécdb % 2) a lécdb 2-vel vett osztási maradéka. A két egyenlőségjel (==) pedig összehasonlító művelet, a (lécdb % 2 == 1) kifejezés tehát akkor igaz, ha a lécdb 2-vel vett osztási maradéka 1, azaz a lécdb páratlan. Egy egyenlőségjelet (=) akkor használunk, amikor a változóknak értéket adunk.[6]

[6] A LibreLogo a Logo hagyományainak megfelelően elfogadja az egy egyenlőségjelet is összehasonlításnál: ha lécdb % 2 = 1, de a jegyzet a Python és egyéb programnyelvekben egyedül használható kettős egyenlőségjelet fogja használni a későbbiekben.

A fenti példában páros lécszám esetén nem kell a teknőcnek rajzolnia, azonban előfordulnak olyan feladatok, amelyeknél a feltétel függvényében kétféle utasítást kell végrehajtanunk. Például a kerítés rajzoló eljárásunkat úgy is megvalósíthatjuk, hogy ha az ismétlésben a hányadik értéke páratlan, akkor barna, különben pedig sárga lécet rajzoljunk ki.

A ha utasítással arra is van lehetőségünk, hogy a feltétel nem teljesülése esetén másik utasítást hajtson végre a program, ezeket az utasításokat a szögletes zárójel után újabb szögletes zárójelek között kell megadnunk.

```
ha feltétel [
    igaz feltétel esetén végrehajtandó utasítások
][
    hamis feltétel esetén végrehajtandó utasítások
]
```

Ennek ismeretében a kerítés feladat így is megvalósítható:

```
eljárás kerítés lécdb
    szín1 = töltőszín
    ismétlés lécdb [
        ha hányadik % 2 == 1 [
            töltőszín „barna”
        ][
            töltőszín „sárga”
        ]
        léc 60
        tollatfel jobbra 90 előre 10 balra 90 tollatle
    ]
    tollatfel jobbra 90 hátra lécdb*10 balra 90 tollatle
    töltőszín szín1
vége
```

Logikai kifejezések

Azt mondtuk, hogy a ha után olyan kifejezések kerülhetnek, amelyek értéke csak igaz vagy hamis lehet. Ezeket a kifejezéseket nevezzük logikai kifejezéseknek. Az alábbi táblázat azt mutatja, hogy milyen relációk segítségével készíthetünk ilyen logikai kifejezéseket. A táblázatban az a és b kifejezések lehetnek konkrét értékek (számok, szövegek), eljárásparaméterek vagy változók (ezekről később lesz szó).

Kifejezés	Jelentése
a == b	A kifejezés igaz, ha a értéke megegyezik b értékével, különben pedig hamis.
a != b	A kifejezés igaz, ha a értéke nem egyezik meg b értékével, különben pedig hamis.
a < b	Számok esetén a kifejezés igaz, ha a értéke kisebb b értékénél, különben pedig hamis. Szöveg esetén a kifejezés igaz, ha az a szöveg megelőzi az ábécé szerinti sorrendben b-t.
a <= b	Számok esetén a kifejezés igaz, ha a értéke kisebb vagy egyenlő b értékénél, különben pedig hamis. Szöveg esetén a kifejezés igaz, ha az a szöveg megelőzi az ábécé szerinti sorrendben b-t, vagy megegyezik vele.
a > b	Számok esetén a kifejezés igaz, ha a értéke nagyobb b értékénél, különben pedig hamis. Szöveg esetén a kifejezés igaz, ha az b szöveg megelőzi az ábécé szerinti sorrendben a-t.
a >= b	Számok esetén a kifejezés igaz, ha a értéke nagyobb vagy egyenlő b értékénél, különben pedig hamis. Szöveg esetén a kifejezés igaz, ha az b szöveg megelőzi az ábécé szerinti sorrendben a-t, vagy megegyezik vele.

A második kifejezéshez közvetlenül tapadó -ban/-ben jelöli a tartalmazás relációt (ez a fejlett adatszerkezeteket bemutató későbbi fejezetekben fog előkerülni):

Kifejezés	Jelentése
a b-ben	A kifejezés igaz, ha az a értékét tartalmazza b. Szövegek esetén az a szövegrészlet előfordul a b szövegben. Ha b lista vagy halmaz, akkor ennek eleme a értéke is. Ha b szótár, akkor a értéke kulcs benne (l. később). Különben pedig hamis a kifejezés. A -ben helyett írható -ban is az olvashatóság céljából, például: b a-ban.

Logikai műveletek

Előfordulhat, hogy több kifejezés együttesen alkot egy feltételt, azaz több kifejezésből valamilyen szabály szerint szeretnénk egy újabb logikai kifejezést készíteni. Ekkor megkülönböztetünk két esetet. Az első esetben azt szeretnénk, hogy egy utasítás akkor hajtódjon végre, ha mindkét feltétel igaz. A másik esetben ele-

gendő az egyik feltétel teljesülése is. Ennek megfelelően két logikai műveletünk van, amellyel két logikai kifejezésből egy újabb logikai kifejezést kapunk. Ezeket az alábbi táblázat tartalmazza, amelyben A és B egy-egy logikai kifejezést jelent.

Logikai művelet	Jelentése
A és B	Az így kapott logikai kifejezés akkor igaz, ha az A kifejezés és a B kifejezés is igaz.
A vagy B	Az így kapott logikai kifejezés akkor igaz, ha az A és B kifejezések közül legalább az egyik igaz.

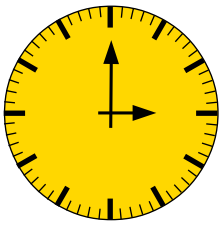
Van még egy logikai műveletünk, amely egyetlen logikai kifejezésből készít egy új logikai kifejezést, amely pontosan akkor igaz, amikor az eredeti kifejezés hamis. Ezt a logikai műveletet az alábbi táblázat mutatja be.

Logikai művelet	Jelentése
nem A	Az így kapott logikai kifejezés akkor igaz, ha az A kifejezés hamis, különben pedig hamis.

Feladatok

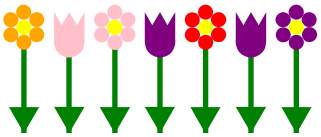
23) Készíts erdő néven eljárást, amely lombos és fenyőfát rajzol felváltva egymás mellé!

24) Készíts órát az alábbi minta szerint, az eljárás paramétere az óralap átmérője legyen!

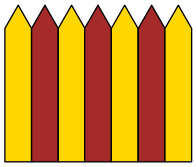


25) Készíts hullám nevű eljárást, amely az alábbi mintának megfelelő hullámot rajzolja meg, az eljárás paramétere a hullámok száma legyen!

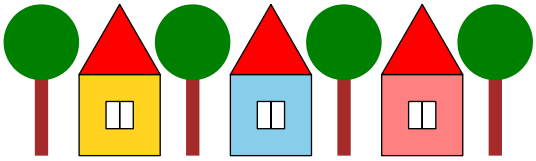
26) Készíts virágoskert nevű eljárást, amely felváltva rajzolja a két típusú virágot!



27) Rajzolj kerítést az alábbi minta szerint, változtatható méretben!



28) Készíts utca néven eljárást, amely felváltva rajzol ki egy-egy fát és házat, az eljárás paramétere a fák és házak együttes száma legyen!



Adatok beolvasása és kiírása

A korábbiakban rajzaink minden paraméterét be kellett írnunk a kódba. Ha szerettük volna más paraméterekkel kipróbálni az eljárásunkat, akkor ehhez át kellett írnunk a kódot. Milyen jó lenne, ha rajzolás közben beszélgethetnénk a teknőccel, és megmondhatnánk neki, hogy most milyen paraméterekkel dolgozzunk.

Valójában egy irányban már kommunikáltunk a teknőccel. Ki tudtuk írni a teknőc állapotait, helyét és irányát:

```
ki hely
ki irány
```

Hasonlóan tudjuk megjeleníteni a teknőc tollának állapotait:

```
ki tollvastagság
ki tollszín
ki tollstílus
ki tollsarok
ki töltőszín
```

A színek RGB kódját írja ki a program.

Hasonlóan írtuk ki az ismétlés szerkezet belső változóját:

```
ismétlés 5 [
    ki hányadik
]
```

A ki utasítással tehát megjeleníthetjük a változók értékét, a teknőc tollállapotait és egyéb állapotait, sőt akár egy eljárás paramétereit, vagy kész szöveget is.

```
eljárás háromszög oldal szín
    ismétlés 3 [
        előre oldal
        jobbra 120
    ]
     töltőszín szín
     tölt
    ki szín
vége
ki „Most egy kitöltött háromszöget fogok rajzolni.”
háromszög 100 „kék”
```

A fenti kis program először kiírja, hogy „Most egy kitöltött háromszöget fogok rajzolni.”, majd megrajzolja a háromszöget, végül kiírja a háromszög színét, amely jelenleg kék.

Most szeretnénk, ha a program futása közben mondanánk meg a teknőcnek, hogy milyen színű háromszöget rajzoljon. Ehhez szükségünk lesz egy változóra, amelyben eltároljuk a felhasználó által megadott színt, legyen ennek a változónak a neve szín. A szín beolvasása és eltárolása a szín nevű változóba a következőképpen történik:

szín = be „Milyen színű legyen a háromszög?”

A beolvasás tehát a be utasítással történik, amely után idézőjelek között kiírunk egy szöveget, ami arra utal, hogy milyen értéket várunk, a kapott választ pedig eltároljuk a szín nevű változóba. Rögtön írjuk ki a szín változó értékét, hogy megbizonyosodjunk arról, hogy sikeres volt a beolvasás:

```
ki szín
```

Sikeres volt! Tehát használhatjuk a szín változót a háromszög kirajzolásánál:

```
szín = be „Milyen színű legyen a háromszög?”
háromszög 100 szín
```

Tehát az imént beolvasott szín változót használhatjuk a háromszög eljárásunk paramétereként.

Rögtön adódik, hogy a háromszög oldalhosszát is a felhasználó adja meg. Egészítsük ki a kódot az oldalhossz beolvasásával!

```
szín = be „Milyen színű legyen a háromszög?”
oldal = be „Mekkora legyen a háromszög oldala?”
háromszög oldal szín
```

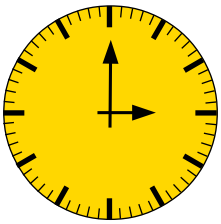
Ekkor azonban az oldalhossz beolvasása után hibaüzenetet kapunk: *„Hiba (3. sor)!”* Azért kaptunk hibaüzenetet, mert a beolvasott érték minden esetben szöveg lesz, azaz például nem 100, mint szám, hanem 100, mint az 1, 0, 0 karakterekből álló szöveg. Ez a szín beolvasásánál nem okozott gondot, mert a színek egyébként is megadhatók szöveggel, de a számoknál probléma. Ezek szerint számértékeket nem is tudunk beolvasni? Szerencsére ez nem így van, csak a szövegként beolvasott számot számmá kell alakítanunk, a feladattól függően egész vagy tört számmá, az egész vagy a tört függvény segítségével. Ennél a feladatnál elég egésszé alakítani a beolvasott értéket.

```
szín = be „Milyen színű legyen a háromszög?”
oldal = egész ( be „Mekkora legyen a háromszög oldala?” )
```

háromszög oldal szín

Feladatok

29) Készíts órát rajzoló eljárást, az órán a felhasználó által megadott időnek (óra, perc) megfelelően jelenjenek meg a mutatók!

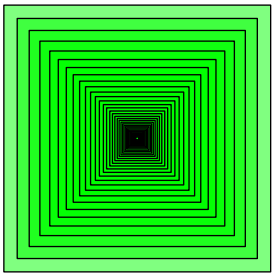


30) Az előző fejezetekben elkészített rajzok programjait alakítsd interaktívvá, azaz olvasd be a rajzok különböző paramétereit!

Rekurzió, fraktálok rajzolása rekurzív eljárások segítségével

Sok programnyelvhez hasonlóan a LibreLogóban is tudunk rekurzív eljárásokat készíteni, azaz olyan eljárásokat, amelyek használják saját magukat. Az ilyen eljárásokat olyan feladatok megoldásában alkalmazzuk, amelyeknél valamilyen szabályszerűségnek megfelelő ismétlés található. Vizsgáljuk meg az alábbi eljárást!

```
eljárás négyzetek méret
    négyzet méret
    négyzetek méret*0,9
vége
négyzetek 200
```



Mit csinál a négyzetek eljárás? Megrajzol egy 200 pont oldalú négyzetet, majd ismét meghívja önmagát 200*0,9=180 paraméterrel, tehát rajzol egy 180 pont oldalú négyzetet, és meghívja önmagát 180*0,9=162 paraméterrel és így tovább.

Az eljárás addig nem áll meg, amíg a *Leállítás* gombbal le nem állítjuk. A teknőc először egyre kisebb és kisebb négyzeteket rajzol, majd amikor már túl kicsi a négyzet oldalhossza, akkor meg is bolondul a teknőc. Mit tehetünk, hogy ezt a végtelen futást megállítsuk? Például azt, hogy csak akkor hívjuk meg újra a négyzetek eljárást, ha a méret még 1-nél nagyobb:

```
eljárás négyzetek méret
    ha méret>1 [
        négyzet méret
        négyzetek méret*0.9
    ]
vége
négyzetek 200
```

A rekurzív eljárások esetén mindig gondoskodnunk kell róla, hogy a rekurzió egyszer leálljon. Ezért minden rekurzív eljárás tartalmaz egy elágazást, amelynek

feltétele a rekurzió során biztosan hamissá válik. A fenti példában a méret paraméter minden rekurzív hívás után 0,9-ére csökken, tehát egy idő után mindenképpen kisebb lesz mint 1. Amikor ez bekövetkezik, akkor véget ér az eljárás.

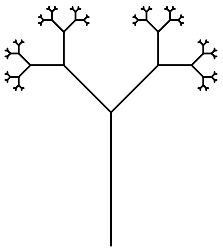
Most biztosan felmerülhet benned a kérdés, hogy mi értelme a rekurziónak, ha a fenti rajzot el tudjuk készíteni az ismétlés utasítás segítségével is. Nos ez valóban így van, a fenti példa valóban megoldható ismétléssel is, ahogy ezt egy korábbi fejezetben meg is csináltuk. Azonban a rekurzív eljárásokkal még sokrétűbb alakzatokat készíthetünk, mint pusztán az ismétlés utasítással. Valamint több programozási feladat hasznos eszköze lehet a rekurzió, például matematikai sorozatok előállításában, rendezési feladatoknál és még sok helyen.

Számunkra a rekurzió az önhasonló alakzatok előállításában nyújt majd segítséget. Ezeket az önhasonló alakzatokat más néven *fraktálok*nak is nevezzük. Azért önhasonlóak, mert egy részletüket tekintve az egészhez hasonló alakzatot kapunk.

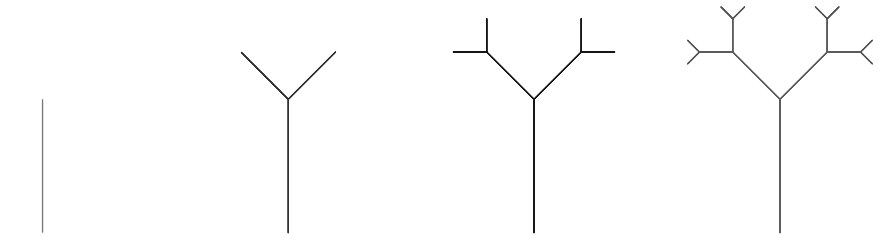
A természetben rengeteg fraktálszerű képződmény található: a fák, a felhők, a páfrányok, a karfiolfajták, a hegyek, a tüdő légcsövecske-rendszere, a csigaházak stb.

Hogy megismerjük a fraktálok készítésének fortélyait, egy fát fogunk rajzolni. A fa minden ága egy fele akkora fa, mint az eredeti, tehát ő egy fraktál!

```
eljárás fa méret szint
    ha szint==0 [
        előre méret
        hátra méret
    ][
        előre méret
        jobbra 45 fa méret/2 szint–1
        balra 90 fa méret/2 szint–1
        jobbra 45
        hátra méret
    ]
vége
fa 50 6
```



Hogy elemezzük a fenti eljárást, nézzük meg, mit rajzol, ha 0, 1, 2, 3 szinttel hívjuk meg.



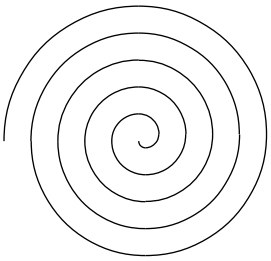
fa 50 0 fa 50 1 fa 50 2 fa 50 3

Ha a szint 0, akkor a fánk csupán egyetlen ágból áll. Ha több szinttel hívjuk meg az eljárást, akkor megrajzoljuk a törzset, illetve ágat, majd balra fordulunk 45 fokot. Ezután meghívjuk a fa eljárást eggyel kevesebb szinttel, fele akkora méretben, majd jobbra fordulunk 90 fokot, és oda szintén egy fele akkora, eggyel kevesebb szintű fát rajzoltatunk ki. Mivel a rekurzív hívásokban mindig eggyel csökken a szintek száma, ezért szükségképpen ez az eljárás véget fog érni.

A fraktálok készítése és a rekurzió egy új gondolkodásmódot igényel, ezért a fejezetben igyekszem minél több feladatot bemutatni, amelyek ismeretében már önállóan is boldogulsz majd a fejezetek végén lévő feladatokkal.

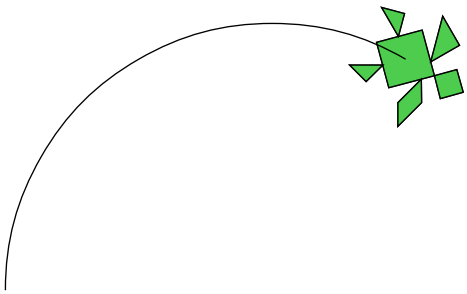
Spirál

Elsőként az alábbi képen látható spirált megrajzoló eljárást készítjük el.



A spirálunk valójában lineárisan növekvő sugarú negyedkörívekből épül fel. Körívdarabokat az ellipszis eljárással tudunk készíteni, azonban a feladatunk megoldását nagyban könnyíti, ha az ellipszis eljárást felhasználva készítünk egy körív eljárást, amelynél a teknőc a köríven haladva rajzol adott sugarú, adott szögű körívet:

```
eljárás körív sugár szög
    tollatfel balra 90 hátra sugár tollatle
    ellipszis [sugár*2, sugár*2, 0, szög, 3]
    tollatfel jobbra szög előre sugár jobbra 90 tollatle
vége
körív 100 120
```



A körív eljárást felhasználva egyre kisebb negyedköríveket rajzolunk. Tehát rajzolunk egy adott méretű negyedkörívet, majd ismét meghívjuk a körív eljárást 5-tel kisebb méretben és eggyel kevesebb szinttel (hiszen egy ívet az imént már kirajzoltunk). Ez alapján a következő eljárást készítettük el:

```
eljárás spirál méret szint
    ha szint != 0 [
        körív méret 90
        spirál méret–5 szint–1
    ]
vége
kép [ spirál 100 20 ]
```

Azonban ha 100 és 20 paraméterekkel hívjuk meg az eljárásunkat, akkor egy idő után már 0 sugarú körívet rajzoltatnánk a teknőccel, ami nem lehetséges. Ezt kiküszöbölhetjük, ha a feltételt kiegészítjük a következő módon:

```
eljárás spirál méret szint
    ha méret > 0 és szint != 0 [
        körív méret 90
        spirál méret–5 szint–1
```

```
    ]
vége
kép [ spirál 100 20 ]
```

Így a rekurzív hívás akkor sem történik meg, ha a méret paraméter negatívvá válik.

Koch-fraktál

A *Koch-fraktál* minden fraktálokról szóló műben megjelenik, ez nem véletlen. Hiszen egy egyszerű, azonban nagyon szép fraktálról van szó. Nézzük meg a Koch-fraktál első néhány szintű változatát:

Koch 100 1 Koch 100 2 Koch 100 3 Koch 100 4

A *Koch-fraktál* tehát úgy készül, hogy veszünk egy szakaszt. A következő szinten a szakaszt elharmadoljuk és a középső harmadának helyére egy szabályos háromszög két másik oldalát rajzoljuk ki. A következő szinteken az előző szinten lévő összes szakaszt elharmadoljuk és a középső harmadokra kerül a szabályos háromszög másik oldalainak vonala.

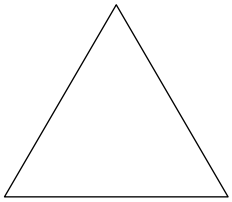
Hogyan lesz ebből rekurzió? Hogy a rekurziót felismerjük, elemeznünk kell a fraktált. A Koch második szintje négy 1-es szintű *Koch-fraktálból* áll, amelyek mérete a 2-es szintű fraktál harmada. Ha megfigyeljük a 3. szintű *Koch*ot, akkor rájövünk, hogy arról is elmondható, hogy négy kettes szintű fraktálból áll, és ugyanolyan elhelyezkedésben, mint a 2-es szintben az egyes szintű fraktálok: vízszintesen, balra 60 fokkal elforgatva, majd jobbra 120 fokkal elforgatva, majd ismét vízszintesen, azaz balra 60 fokkal forgatva.

Nézzük tehát a Koch-fraktál megrajzoló rekurzív eljárást:

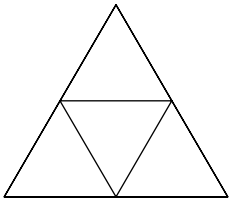
```
eljárás Koch méret szint
    ha szint==1 [
        előre méret
    ][
        Koch méret/3 szint–1 balra 60
        Koch méret/3 szint–1 jobbra 120
        Koch méret/3 szint–1 balra 60
        Koch méret/3 szint–1
    ]
vége
```

Sierpiński-háromszög

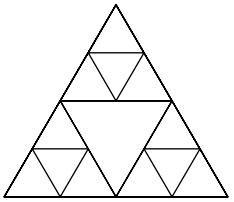
Egy másik híres fraktál a *Sierpiński-háromszög*, amelyet úgy kapunk, hogy egy háromszöget négy kisebb háromszögre bontunk az oldalfelező pontjait összekötő szakaszok (középvonalak) segítségével. Az így kapott alakzatból a középső háromszöget „kivágjuk”, majd a megmaradt 3 háromszögből megint kivágjuk a középvonalaik által határolt kis háromszögeket, és így tovább. A *Sierpiński-háromszög* első néhány példányát az alábbi táblázat mutatja.



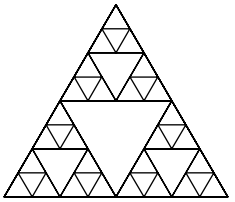
Sierpiński 90 1



Sierpiński 90 2



Sierpiński 90 3



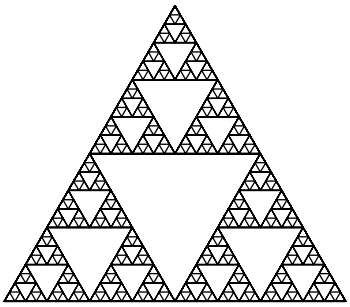
Sierpiński 90 4

A példánkban olyan *Sierpiński-háromszöget* készítünk, amelynek csak a körvonalait rajzoltuk meg. A mintákat megfigyelve a következő rekurziót állapíthatjuk meg. A *Sierpiński-háromszöget* úgy kapjuk, hogy egy háromszöget a középvonalaival (oldalfelezőpontokat összekötő szakaszokkal) négy háromszögre bontjuk és ezek közül elhagyjuk a középsőt, a maradék három háromszöggel ugyanezt az eljárást hajtjuk végre. Tehát az eljárás a következőképpen valósítható meg:

```
eljárás Sierpiński méret szint
    ha szint !=0 [
        ismétlés 3 [
            Sierpiński méret/2 szint–1
            előre méret jobbra 120
        ]
    ]
vége
```

A *Sierpiński-háromszög* egy magasabb szintű példánya:

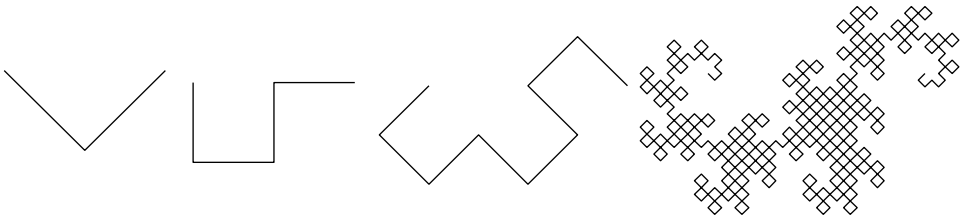
```
Sierpiński 128 6
```



Sárkánygörbe

A *sárkánygörbe* szintén egy híres görbe (még a Jurassic Park című filmben is elhangzik a neve). Valószínűleg azért nevezik sárkánygörbének, mert elég nagy lépésszám esetén egy sárkányra emlékeztet.

A sárkánygörbe előállítása a következő. A kiindulási alakzat egy négyzet sarka, amelynek szakaszait a következő lépésben egy-egy olyan négyzet sarkával helyettesítünk, amelynek átlója lenne az eredeti szakasz (oldalhossza $\sqrt{(2)}$-ed része a helyettesített szakaszának). Azonban a két négyzetsarok ellentétesen fordul, emiatt nemcsak méret és szint paramétere lesz az eljárásunknak, hanem fordul paramétere is.



sárkány 90 1 1 sárkány 90 1 2 sárkány 90 1 3 sárkány 90 1 9

Nézzük a *sárkánygörbe* librelogós megvalósítását:

```
eljárás sárkány méret fordul szint
    ha szint==0 [
        előre méret
    ][
        jobbra 45*fordul
        sárkány méret/gyök 2 1 szint–1
        balra 90*fordul
        sárkány méret/gyök 2 –1 szint–1
        jobbra 45*fordul
```

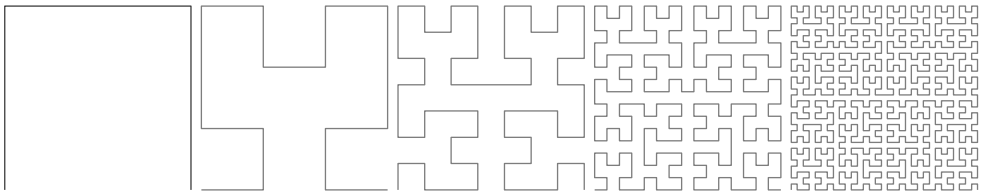
```
  ]
vége
```

Megjegyzés: A példában a gyök 2 *függvény hívás eredményeként kapjuk meg a 2 négyzetgyökét.*

Ennél a megoldásnál a teknőc a fraktál bal szélén kezdi a rajzolást, és egy vonallal készíti el a sárkánygörbét. A második „sarok” mindig az ellenkező irányba kanyarodik, ezért kell a fordul paramétert (–1)-gyel szorozni a rekurzív híváskor (jobbra –90 = balra 90).

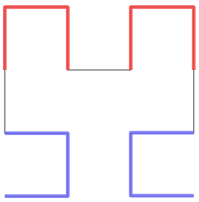
Hilbert-görbe

A Hilbert-görbe szintén egy olyan görbe, amelynél szükségünk van a fordulás irányának változtatására. A görbe azért különleges, mert ha végtelenített eljárással készítjük el, akkor a görbe az egész síkfelületet kitölti. A táblázat a fraktál első néhány szintű változatát tartalmazza.



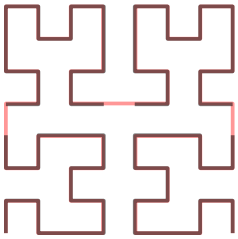
Hilbert 100 1 1 Hilbert 100 1 2 Hilbert 100 1 3 Hilbert 100 1 4 Hilbert 100 1 5

Mielőtt a fraktál eljárásának kódolásába kezdenénk, elemezzük egy kicsit a görbe szerkezetét, hogyan jelennek meg a korábbi szintek a magasabb szintű görbében:



A rajzolást most is úgy képzeljük, hogy a görbe bal alsó sarkából indul a teknőc, és vízszintes irányba néz. A 2-es szintű Hilbert-görbén pirossal jelöltem a görbe azon részét, ahol az 1-es szintű Hilbert-görbe jobbra fordulással; és kékkel, ahol balra fordulással áll elő. Tehát a görbénk úgy készül, hogy van egy balra forduló kisebb szintű görbénk, haladunk valamennyit, majd van egy jobbra forduló görbénk, megint haladunk, megint jobbra forduló görbe, haladunk és végül egy

balra forduló kisebb szintű görbe következik. De mennyit kell haladnunk a két kisebb szintű görbe között? A második szintet vizsgálva azt mondanánk, hogy annyit, amekkora az eggyel kisebb szintű görbék méret paramétere. Azonban ez a harmadik szintnél nem igaz.



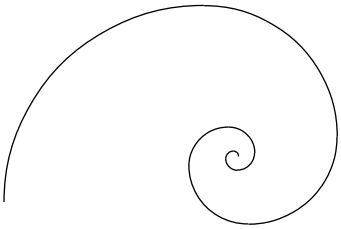
Valójában minden szinten annyit kell haladni, mint a legkisebb szintű alakzat mérete. Ha minden lépésben harmadoljuk a görbe méretét, akkor a haladás mértékét a méret/3^{szint} képlettel kapjuk meg. Tehát az eljárásunk a következőképpen alakul:

```
eljárás Hilbert méret fordul szint
    ha szint==0 [
        balra 90*fordul
        ismétlés 3 [ előre méret jobbra 90*fordul ]
        jobbra 180
    ][
        balra 90*fordul
        Hilbert méret/3 (–1)*fordul szint–1
        előre méret/3**szint jobbra 90*fordul
        Hilbert méret/3 fordul szint–1
        előre méret/3**szint
        Hilbert méret/3 fordul szint–1
        jobbra 90*fordul előre méret/3**szint
        Hilbert méret/3 (–1)*fordul szint–1
        balra 90*fordul
    ]
vége
```

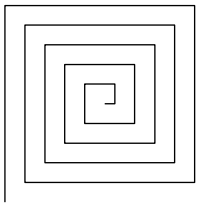
Most, hogy ennyi szép fraktált megismertél, arra biztatlak, hogy a kész eljárásokkal kísérletezz egy kicsit. Változtasd meg az eljárások különböző paramétereit, és figyeld meg a fraktál változásait. Lehet, rátalálsz egy olyan fraktálra, amelyet előtted még senki nem ismert! A feladatok között találsz további érdekes fraktálokat. Ha ezekkel is megbirkóztál, nyugodtan kutass az interneten további érdekes rekurzív alakzatok után.

Feladatok

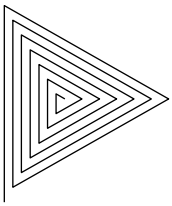
31) A bemutatott spirál mintájára készítsd el az alábbi spirálokat!



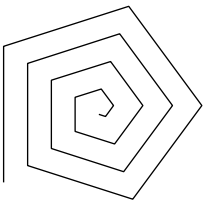
spirál 1 90 10



spirál 2 90 20

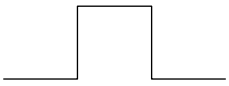


spirál 3 90 20

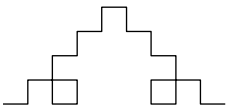


spirál 4 50 20

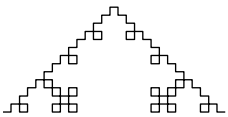
32) A *Koch-fraktálnak* van egy négyszögletes változata is, amelynek az első néhány szintjét az alábbi táblázat mutatja. Készíts rekurzív eljárást a négyszögletes *Koch-fraktál* megrajzolására!



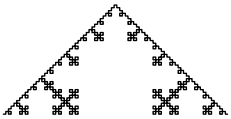
Koch2 100 1



Koch2 100 2



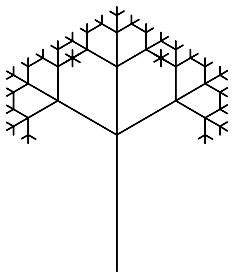
Koch2 100 3

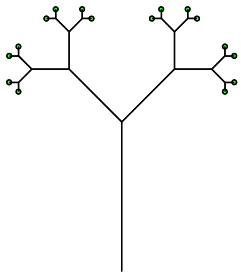


Koch2 100 4

33) A fa fraktálnak sok különféle változata létezik, készítsd el a következő fákat megvalósító rekurzív eljárásokat!

a) Háromágú fa (bal oldalon):

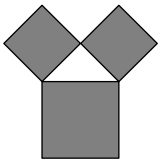




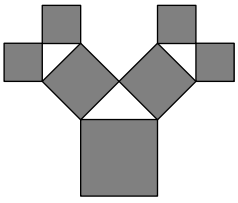
b) Bogyós fa (jobbra fent);

c) Pitagorasz-fa:

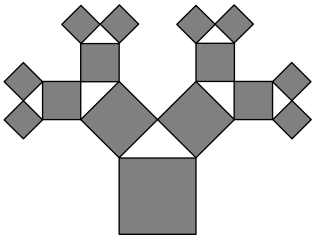
Pitagorasz 30 1



Pitagorasz 30 2

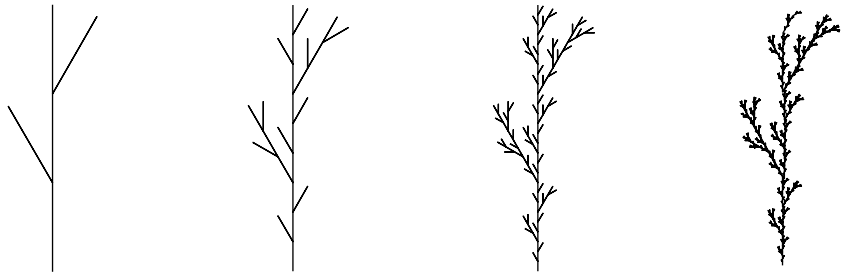


Pitagorasz 30 3



Pitagorasz 30 4

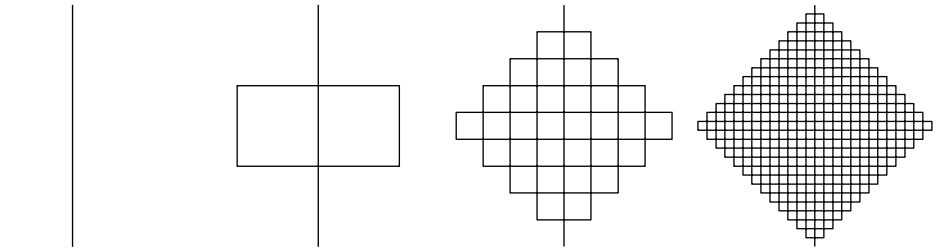
d) Aszimmetrikus fa:



fa2 100 1 fa2 100 2 fa2 100 3 fa2 10 0 4

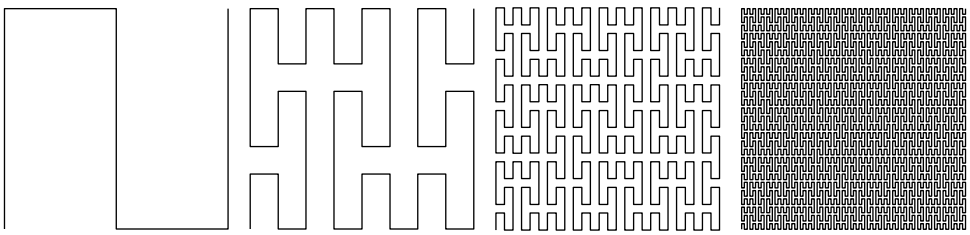
34) Peano nevéhez több fraktál is fűződik, ezek közül kettőt mutatok most be. Készítsd el az őket megrajzoló rekurzív eljárásokat!

a) Peano1



Peano1 5 0 1 Peano1 5 0 2 Peano1 5 0 3 Peano1 5 0 4

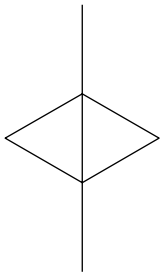
b) Peano2



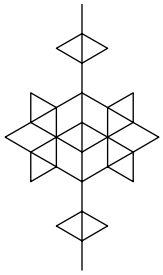
Peano2 50 1 Peano2 50 2 Peano2 50 3 Peano2 50 4

35) Peano1 és a Koch-fraktál ötvözete:

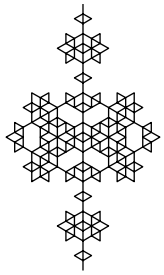
Peano_Koch 5 0 1



Peano_Koch 5 0 2



Peano_Koch 5 0 3

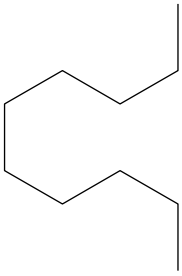


Peano_Koch 5 0 4

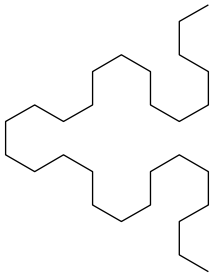
36) További *Sierpiński-fraktálok*:

a) Nyílhegy

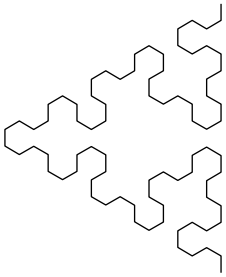
nyílhegy 50 1



nyílhegy 50 2

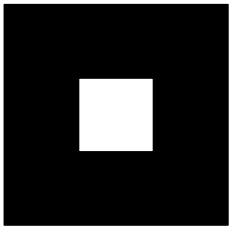


nyílhegy 50 3

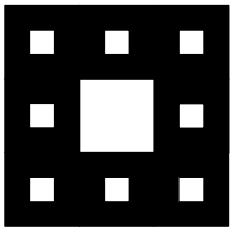


nyílhegy 50 4

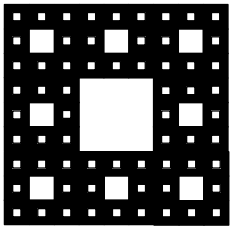
b) Szőnyeg



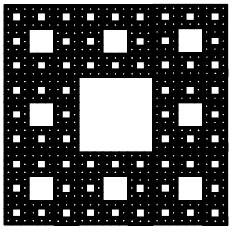
szőnyeg 5 0 1



szőnyeg 5 0 2

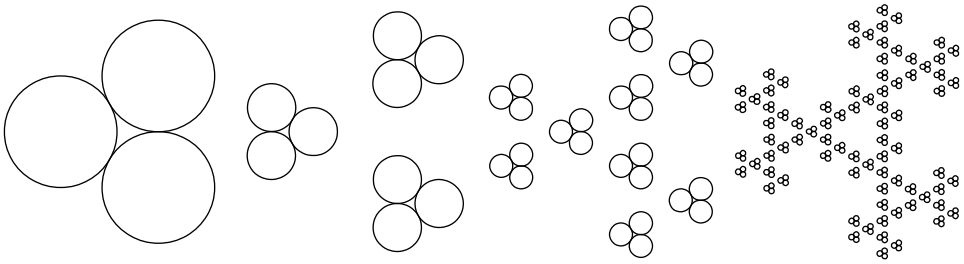


szőnyeg 5 0 3



szőnyeg 5 0 4

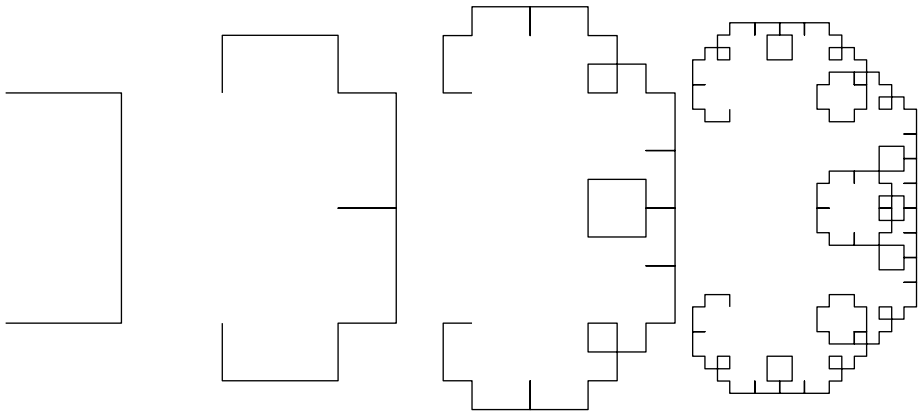
37) Körök



körök 5 0 2 | körök 5 0 3 | körök 5 0 4 | körök 5 0 6

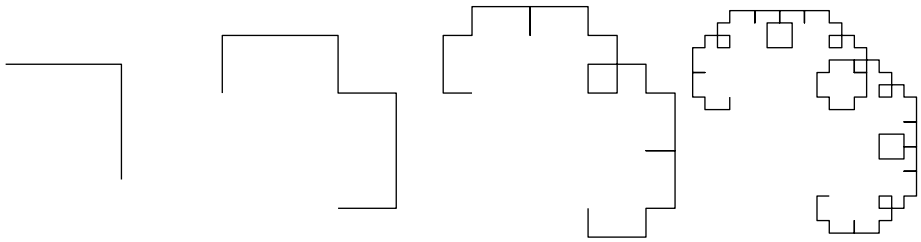
38) *Levy-fraktálok*

a) Levy1



Levy1 100 1 | Levy1 100 2 | Levy1 100 3 | Levy1 100 4

b) Levy2



Levy 2 5 0 1 | Levy 2 5 0 2 | Levy 2 5 0 3 | Levy 2 5 0 4

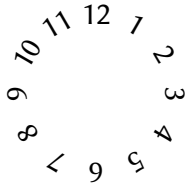
Szövegdobozok használata, betűszín, betűstílus beállításai

A rajzainkat szövegekkel is kiegészíthetjük a címke utasítás segítségével. Az alábbi utasításszerkezet a LibreLogo szöveget írja a képernyőre egy szövegdobozba. A szövegdoboz a teknőc helyére kerül az irányának megfelelően.

```
címke „LibreLogo”
```

A címke utasítással nem csak kész szöveget, de egy változó értékét is kiírhatjuk:

```
ismétlés 12 [
    tollatfel jobbra 30 előre 30 tollatle
    címke hányadik
    tollatfel hátra 30 tollatle
]
```



Ennek segítségével már készíthetünk például számozott óralapot, lásd a fejezet végi feladatokat!

Természetesen beolvasott változó értékét is kiírhatjuk. Próbáld ki a következő kódot!

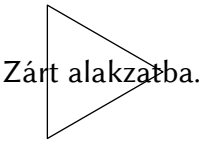
```
név = be „Hogy hívnak?”
címke név
```

A LibreLogóban nem csak szövegdobozokba írhatunk ki szöveget, hanem az éppen megrajzolt alakzataink belsejébe is.

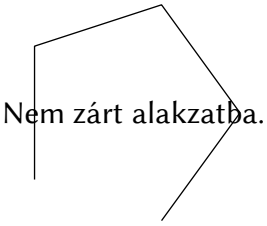
```
négyzet 100
szöveg „Egy négyzetbe.”
```

ellipszis [100, 40]
szöveg „Egy ellipszisbe.”

ismétlés 3 [előre 50 jobbra 120] zár
szöveg „Zárt alakzatba.”



ismétlés 4 [előre 50 jobbra 72]
szöveg „Nem zárt alakzatba.”



A mintákból láthatjuk, hogy szöveget készíthetünk kitöltött és zárt alakzatok belsejébe, vagy nem zárt alakzatokba, a szöveg ekkor mindenképp vízszintesen jelenik meg az alakzat közepén.

Mind a szövegdobozba, mind az alakzatok belsejébe írt szövegeink tulajdonságait módosíthatjuk. A szövegtulajdonságokat, ahogy korábban a toll tulajdonságok esetén is, a szöveg kiírása előtt kell beállítanunk. Ameddig az egyes tulajdonságokat nem állítjuk vissza, addig a következő szövegeink megtartják a tulajdonságaikat. Ezek szemléltetésére íme néhány példa:

betűszín „kék”
címke „Kék színű címke.”

Kék színű címke.

ismétlés 4 [előre 50 jobbra 90]
szöveg „Kék színű alakzat szöveg.”

betűméret 20
címke „20 pontos szöveg.”

20 pontos szöveg.

betűméret 12
címke „12 pontos szöveg.”

12 pontos szöveg.

betűcsalád „Linux Libertine G”
címke „Linux Libertine G betűs szöveg.”

Linux Libertine G betűs szöveg.

betűvastagság „kövér”
címke „Félkövér szöveg.”

Félkövér szöveg.

betűvastagság „normál”
címke „Ez ismét egy normál vastagságú szöveg.”

Ez ismét egy normál vastagságú szöveg.

betűstílus „dőlt”
címke „Dőlt betűs szöveg.”

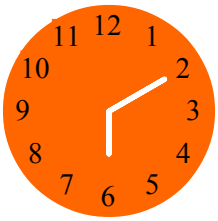
Dőlt betűs szöveg.

betűstílus „álló”
címke „Ez ismét egy nem dőlt szöveg.”

Ez ismét egy nem dőlt szöveg.

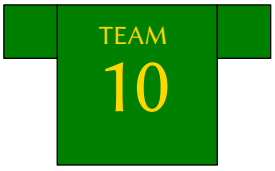
Feladatok

39) Készíts a mintának megfelelő számozott óralapot, amely a felhasználó által megadott óra és perc értékeknek megfelelő időt mutatja!

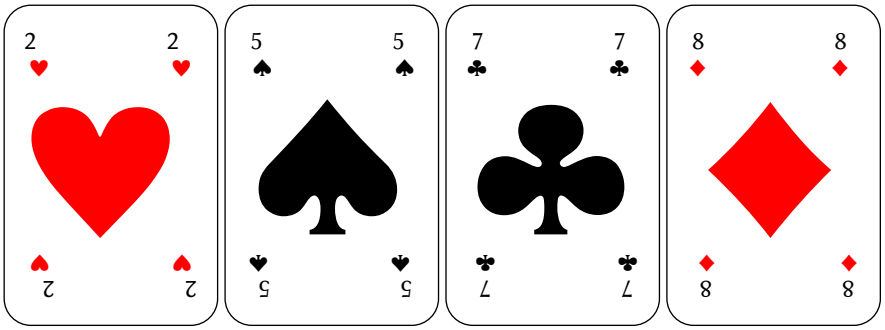


40) Készíts ötszöget, amelynek oldalai a LibreLogo szövegből állnak!

41) Készíts mez néven eljárást, amelynek paraméterei a mez mérete, a színe, a felirat színe, a csapat neve és a mez száma!



42) Készítsd el az alábbi kártyalapokat a most tanult utasítások segítségével (a kártyák szimbólumait megtalálod a Beszúrás menü Különleges karakterek között)!



43) Készíts névjegykártyát rajzoló eljárást! A nevet és az elérhetőségi adatokat a felhasználó adja meg. A névjegykártyán szerepeljen egy különleges karakterből álló logó, mint például az ábrán látható, DejaVu Sans betűkészletben megtalálható macskafej.

Listák, hivatkozás listaelemekre

Valójában már eddig is találkoztál listákkal a színbeállító, valamint a téglalap és ellipszis eljárások paramétereiként. A listákban mindenféle elemeket sorolhatunk fel, amelyekre aztán a sorszámukkal hivatkozhatunk. A lista elemei lehetnek számok, szövegek, sőt akár bonyolultabb adatok, listák, halmazok, szótárak is. Először az egyszerűbb listákkal ismerkedünk meg, amelyek számokat és szövegeket tartalmaznak csak.

A téglalap és ellipszis eljárások paramétere tehát egy-egy lista. A listák ezek szerint szögletes zárójelek között felsorolt elemek:

[100, 50]
[„alma”, „körte”, „szilva”]

A lista elemeit vesszővel és egy szóközzel választjuk el egymástól. Fontos, hogy a lista első eleme közvetlenül a szögletes zárójel után, utolsó eleme közvetlenül a szögletes zárójel előtt áll. Ezzel különböztetjük meg az elágazásnál és ismétlésnél használt szögletes zárójeles utasításlistától a valódi listákat. A szöveges listaelemeket mindig idézőjelek közé kell tenni, a számokat sosem.

A listáinknak legtöbbször nevet is adunk, ekkor valójában egy változó segítségével tároljuk a listánkat:

számok = [1, 2, 3, 4, 5, 6]

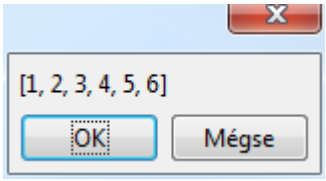
A listanevek megadásakor ugyanazokat a szabályokat kell betartanunk, mint a változóneveknél. Azaz nem tartalmazhatnak szóközt, műveleti jeleket, nem kezdődhetnek számmal és különbözniük kell az eljárásaink, utasításaink neveitől.

Létrehozhatunk üres listát is:

üreslista = []

A lista elemeit megjeleníthetjük üzenetablakban:

ki számok

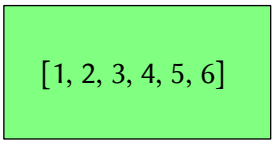


És kiírhatjuk a dokumentumban is:

```
címke számok
```

[1, 2, 3, 4, 5, 6]

```
téglalap [100, 50]
szöveg számok
```



A lista elemeit 0-tól (elemszám−1)-ig számozzuk. Az adott sorszámú elemet a lista[sorszám] szerkezettel kaphatjuk meg:

```
címke számok[0]      ; a számok lista első eleme
címke számok[1]      ; a számok lista második eleme
```

A fenti példában a pontosvessző utáni rész csak megjegyzés, a teknőc figyelmen kívül hagyja, nem akarja végrehajtani. Ilyen megjegyzéseket gyakran használnak a programozók emlékeztetőül, hogy érthetőbb legyen a kód.

A lista utolsó elemét a −1 sorszámmal is megkaphatjuk:

```
címke számok[-1]     ; a számok lista utolsó eleme
címke számok[-2]     ; a számok lista utolsó előtti eleme
```

A listánkhoz adhatunk új elemeket a + művelet segítségével:

```
l = [„első”, „második”]
új = „harmadik”
l = l + [új]
címke l
```

['első', 'második', 'harmadik']

Az összeadás művelettel valójában tetszőleges listákat egymás után fűzhetünk:

```
címke l + számok
```

['első', 'második', 'harmadik', 1, 2, 3, 4, 5, 6]

A listáink elemeit sorba is rendezhetjük a rendez utasítás segítségével. A rendez utasítás a számokat nagyság szerint növekvő, a szövegeket pedig ábécé sorrendbe rakja. A rendez utasítás eredménye a rendezett lista, a lista eredeti sorrendjét nem változtatja meg.

```
K = [2, 4, 1, 5, 2]
ki rendez K
```

[1, 2, 2, 4, 5]

```
ki K
```

[2, 4, 1, 5, 2]

Ha azt szeretnénk, hogy a listánk valóban rendezetté váljon, akkor a rendezett listát el kell tárolnunk az eredeti lista helyén:

```
K = rendez K
ki K
```

[1, 2, 2, 4, 5]

Előállíthatunk egész számokból álló, 0-tól kezdődő, egyesével növekvő listát:

```
T = sor 10          ; T = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Vagy két egész szám közé eső számsorozatot:

```
T1 = sor 10 15      ; T1 = [10, 11, 12, 13, 14]
```

A sorozat lépésközét is beállíthatjuk egy harmadik paraméterrel:

```
T3 = sor 10 20 3 ; T3 = [10, 13, 16, 19]
```

A lista elemszámát a darab utasítással kaphatjuk meg:

```
ki darab T3         ; eredménye: 4
```

Megismertük, hogy néznek ki a listák, megismertünk néhány listaműveletet, de arról még nem esett szó, hogy mire is valók a listák.

Listák alkalmazása

Most néhány olyan feladatot fogok mutatni, amelyek megvalósítása listák segítségével könnyebb, mint nélkülük.

Feladat: Olvassunk be egy egyjegyű számot számként (pl. 2) és írjuk ki a beolvasott számot szövegesen (pl. kettő)!

A feladat megoldásához két segédlistát fogunk használni:

```
egyesek = [„nulla”, „egy”, „kettő”, „három”, „négy”, „öt”, „hat”, „hét”, „nyolc”, „kilenc”]
```

Az egyesek lista megfelelő sorszámú helyén tehát a szám szöveges változata szerepel. Így például az egyesek[3] a „három” szöveg lesz. Tehát ha beolvassuk az egyjegyű számunkat egy szám nevű változóba, akkor a lista szám-adik elemét elegendő kiírnunk. A beolvasáskor természetesen figyelnünk kell a típuskonverzióra!

```
szám = egész(be „Adj meg egy egyjegyű egész számot!”)
```

```
ki egyesek[szám]
```

Azonban ez a program hibát jelez, ha a számunk nem egyjegyű, mert olyan listaelemet akarunk kiírni, amely nincs. Ha pedig negatív egyjegyű számot adunk meg, akkor kiír ugyan szöveget, de −1-nél a „kilenc”, −2-nél a „nyolc” stb. szövegeket.

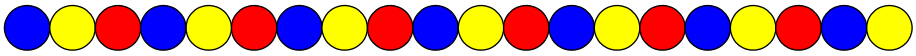
Mindkét problémát kiküszöbölhetjük, ha létrehozzuk az elfogadott értékek, azaz a 0-tól 9-ig terjedő számok listáját a már megismert sor utasítással, majd a számátalakítás feltételévé tesszük, hogy a be utasítással bekért számot tartalmazza-e a 0–9 számokat tartalmazó lista. Ez utóbbit a logikai kifejezéseknél említett -ban/-ben relációval fogjuk megvizsgálni:

```
egyesek = [„nulla”, „egy”, „kettő”, „három”, „négy”, „öt”, „hat”, „hét”, „nyolc”, „kilenc”]
egyjegyűegészek = sor 10
szám = be „Adj meg egy egyjegyű egész számot!”
ha szám egyjegyűegészek-ben [
    ki egyesek[szám]
]
```

Most nézzünk meg egy olyan feladatot is, amelyben rajzolunk is a listák segítségével.

Feladat: Készítsünk sorminta eljárást, amely a felhasználó által megadott színek alapján, megadott darabszámszor periodikusan ismétlődő körökből álló sormintát rajzol!

Például: Ha a megadott színek a piros, kék és a sárga, és 20 elemből álló sormintát szeretnénk rajzolni, akkor a következőt kapjuk:



Valahány elemenként ismétlődő sormintát elágazások nélkül listával oldhatunk meg a legegyszerűbben. Különösen igaz ez, ha előre nem ismert a periódus hossza, azaz, hogy hányféle színnel készítjük a sormintát. A feladat egy lehetséges megoldása:

```
eljárás sorminta méret színek hossz
    ismétlés hossz [
        periódus = darab színek ; a periódus hossza a színek lista elemszáma
        töltőszín színek[hányadik % periódus]
        kör méret
        tollatfel jobbra 90 előre méret balra 90 tollatle
    ]
vége
```

```
színek = [] ; üres listát hoztunk létre
periódus = egész(be „Hány színt szeretnél?”)
ismétlés periódus [
    szín = be „Add meg a következő színt”
    színek = színek + [szín] ; az új színt hozzáfűzzük a színek listához
]
hossz = egész(be „Hossz?”)
kép [ sorminta 20 színek hossz ]
```

Ebben a megoldásban kihasználjuk az ismétlés szerkezet hányadik változóját, a hányadik változó periódushossz szerinti osztási maradéka adja meg, hogy a színek lista hányadik színe lesz a teknőc töltőszíne, azaz milyen színű lesz a következő alakzat.

Feladatok

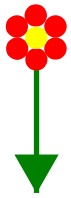
44) Ha ügyesen módosítod az első példafeladat kódját, akkor könnyen megoldhatod az egyjegyű egész számokat szöveggel kiíró eljárást, amely a negatív számokat is ki tudja írni.

45) Készíts eljárást, amely egy kétjegyű pozitív egész számot szöveggel ír ki a képernyőre!

46) Készíts eljárást, amely a felhasználó által megadott méretekkel periodikusan ismétlődő sormintát rajzol, például hópelyhekből!

47) Készíts univerzális kártyalap eljárást, amely paraméterként kapja meg, hogy milyen színű és számú francia kártya lapot kell kirajzolni!

48) Készíts virágrajzoló eljárást, amelynek paramétere egy olyan lista, amelynek első eleme a virág színét, a második eleme a virág méretét – például [„piros”, 40] – tartalmazza!



Véletlen

A véletlen fogalma nagyon fontos a programozásban. Vannak véletlent használó algoritmusok, amelyek hatékonyan oldanak meg nagyon nehéz problémákat, nagyon nagy valószínűséggel, vagy akár biztosan jól. A véletlen fogalma elengedhetetlen a különböző szimulációs feladatokban is. Természetesen mi még keveset tudunk a nagyon nehéz problémák megoldásához, azonban a véletlent a rajzaink készítése során is tudjuk használni, hasonló módon, mint azt a szimulációk során használhatják.

Véletlen számok használata

A teknőcünk a véletlen utasítással véletlen számokat tud előállítani. Ha a véletlen paramétere egy pozitív szám, akkor egy ennél kisebb, de nullánál nagyobb vagy egyenlő véletlen törtszámot fogunk visszakapni.

```
ki véletlen 100
```

A fenti utasítás eredménye egy 0-nál nagyobb vagy egyenlő, és egy száznál kisebb törtszámot ad eredményül. Ha a kapott véletlen számmal a későbbiekben dolgozni szeretnénk, akkor el kell tárolnunk egy változóban.

```
szám = véletlen 10
ki szám
```

Így a szám változóban eltároltunk egy 0 és 10 közé eső törtszámot, amelyet a véletlen utasítás állított elő.

Mit tegyünk, ha nem tört, hanem egész véletlen számot szeretnénk előállítani. Ekkor az adatbeolvasásnál már megismert egész függvényt fogjuk használni, amely a véletlen utasításból kapott törtszámot egész számmá alakítja úgy, hogy a törtrészt levágja a szám végéről.

```
szám = egész véletlen 10
ki szám
```

A szám változó értéke most egy 0 és 9 közé eső egész szám.

Hogyan kaphatunk 200 és 299 közé eső számokat a véletlen utasítás segítségével? Először előállítunk egy 0 és 100 közötti véletlen számot. Ha ehhez hozzáadunk 200-at, akkor 200 és 300 közé eső véletlen számot kapunk. Ha egész számokat szeretnénk kapni, akkor az egész függvényt használva egésszé alakítjuk.

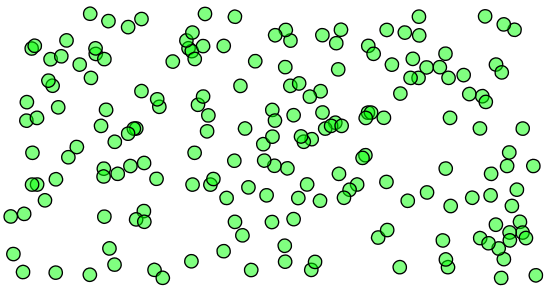
```
szám1 = 200 + véletlen 100
```

```
ki szám1
szám2 = 200 + egész véletlen 100
ki szám2
```

Most már tudunk véletlen számot előállítani. De mire jó ez nekünk? Erre nézzünk meg egy példát.

Feladat: A rajzlapon egy paraméterként megadott méretű téglalapot töltsünk ki véletlenszerűen körökkel!

```
eljárás körök oldal1 oldal2
ismétlés [
    közép = hely
    vízszintes = közép[0]+ egész véletlen oldal1
    függőleges = közép[1]+egész véletlen oldal2
    tollatfel hely [vízszintes, függőleges] tollatle
    kör 5
    tollatfel hely közép tollatle
]
vége
körök
```



Az eljárásunk elején eltároljuk a közép nevű változóba a teknőc aktuális helyét. Majd kiszámoljuk a véletlenszerű vízszintes, majd függőleges pozícióját, amelyet úgy kapunk, hogy a jelenlegi vízszintes pozíciójához hozzáadunk egy véletlen számot, amely legfeljebb a téglalap vízszintes oldalhossza lehet, ugyanezt tesszük a függőleges pozíció esetén is. Majd a hely utasítás segítségével ebbe a véletlenszerű pozícióba küldjük a teknőcöt, kirajzoljuk a kört, majd visszaküldjük a kiinduló pozícióba. Ezeket a lépéseket ismételjük az eljárás során. Mivel nem adtunk meg ismétlésszámot, a teknőc addig rajzolgat köröket, ameddig a leállítás gombbal meg nem állítjuk.

Az eljárásban szereplő módszert, amellyel a teknőc véletlenszerű helyzetét határoztuk meg, használják különböző természettudományos (pl. fizikai, biológiai, kémiai) vagy társadalomtudományos szimulációs feladatok megoldása során is.

Ilyen szimulációk lehetnek például a gázmolekulák mozgását, vagy valamely állatpopuláció összetételének változásait modellező szimulációk.

Véletlen listaelem kiválasztása

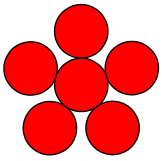
A véletlen számoknál jóval egyszerűbben használhatjuk a véletlen listaelemeket a rajzainkban. A véletlen utasításra van csak szükségünk, amelynek paramétere most egy lista. Ebben az esetben a lista egy véletlenszerű elemét kapjuk eredményül.

```
színek = [„piros”, „kék”, „sárga”, „rózsaszín”, „lila”, „narancs”, „arany”]
ki véletlen színek
```

Feladat: Készíts eljárást, amely megadott számú virágot rajzol egymás mellé, a virágok színe véletlenszerűen piros, kék, rózsaszín, lila, narancssárga vagy aranyszínű legyen!

Készítsünk egy segédeljárást, amely megrajzolja a virágot!

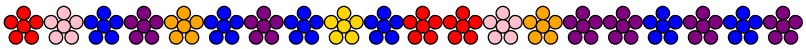
```
eljárás virág méret szín
    töltőszín szín
     ismétlés 5 [
        tollatfel előre méret tollatle
        kör méret
        tollatfel hátra méret tollatle
        jobbra 72
    ]

    kör méret
vége
kép [ virág 20 „piros” ]
```



Most pedig jöjjön a virágokból álló sorminta!

```
eljárás sorminta hossz méret
    színek = [„piros”, „kék”, „rózsaszín”, „lila”, „narancs”, „arany”]
    ismétlés hossz [
        szín = véletlen színek
        virág méret szín
        tollatfel jobbra 90 előre méret*3 balra 90 tollatle
```

```
    ]
vége
kép [ sorminta 20 5 ]
```



A véletlen listaelem segítségével véletlen egész számokat a következőképpen is előállíthatunk:

```
ki véletlen sor 10 30
```

A fenti kód egy 10 és 29 közötti egész számot ad eredményül. Hiszen a sor utasítás egy 10 és 29 közötti számokból álló listát állít elő, amelyből a véletlen utasítás választ egyet véletlenszerűen.

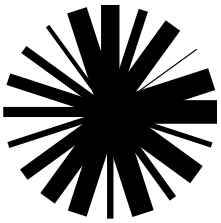
Véletlen paraméterek előállítása

A tetszőleges utasítással a teknőc tollparamétereit tudjuk beállítani véletlenszerűen. Például a töltőszínt:

```
ismétlés 10 [
    töltőszín tetszőleges
    négyzet 10
    tollatfel jobbra 90 előre 10 balra 90 tollatle
]
```

Vagy a tollvastagságot:

```
ismétlés 20 [
    tollvastagság tetszőleges
    előre 40 hátra 40
    jobbra 18
]
```



Feladatok

49) Állítsunk elő a teknőccel 100 és 200 közé eső páros számot véletlenszerűen!

a) Használd a véletlen utasítást számparaméterrel!

b) Használd a sor és a véletlen utasításokat!

50) Készíts véletlen listaelemet előállító eljárást a véletlen utasítás számparaméteres hívását felhasználva!

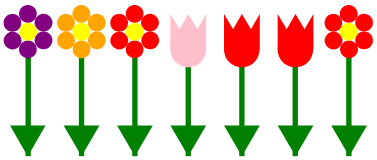
51) Készíts könyvespolcrajzoló eljárást, amely a paraméterként megadott méretű polcra rajzol véletlenszerű vastagságú és színű könyveket!

52) Tölts ki a rajzlapon egy téglalapot véletlenszerűen véletlenszerű méretű hópelyhekkel!

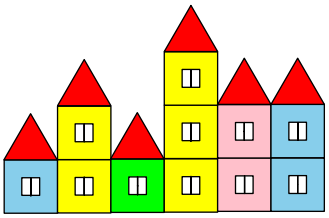
53) Szimuláljunk kockadobást, és rajzoljuk ki a dobókocka megfelelő oldalát!

54) Szimuláld a francia kártya pakliból való véletlenszerű laphúzást, a húzott lapot rajzold ki a képernyőre!

55) Készíts virágoskertet rajzoló eljárást, a virágok színe és fajtája legyen véletlenszerű!



56) Készíts sorházat rajzoló eljárást, a házak magassága és színe legyen véletlenszerű!



Listaelemeken végiglépkedő ciklus

A *Listák, hivatkozás listaelemekre* fejezetben megismerkedtünk a listákkal, valamint néhány alkalmazásukkal. Most tovább bővítjük a listákkal kapcsolatos ismereteinket.

Ha listákkal dolgozunk, akkor nagyon sokszor előfordul, hogy a listaelemeket egyesével kell megvizsgálnunk. Vegyük például a következő feladatot.

Feladat: Pisti nagyon szereti a filmeket, a kedvenceit DVD-n is megvásárolja, amelyeket egy külön polcon tart. A DVD-kről nyilvántartást vezet egy listában. A lista alapján rajzoljuk meg Pisti DVD-s polcát!

```
filmek = [„Gyűrűk ura 1.”, „Gyűrűk ura 2.”, „Gyűrűk ura 3.”, „Hobbit”, „Gru”, „Gru 2”, „Jurassic Park”, „Avatar”, „Hupikék törpikék”, „Hupikék törpikék 2”]
```

Ahhoz, hogy meg tudjuk rajzolni a DVD-ket, minden listaelemre szükségünk van szépen sorban. Ha tudjuk, hány listaelem van, akkor az ismétlés utasítás segítségével végiglépkedhetünk a listán. A listaelemek számát a darab utasítás segítségével kaphatjuk meg, ahogy azt a listákról szóló fejezetben megmutattuk.

```
filmek = [„Gyűrűk ura 1.”, „Gyűrűk ura 2.”, „Gyűrűk ura 3.”, „Hobbit”, „Gru”, „Gru 2”, „Jurassic Park”, „Avatar”, „Hupikék törpikék”, „Hupikék törpikék 2”]
ismétlés darab filmek [
    ki filmek[hányadik–1]
]
```

Ugye emlékszünk, hogy az ismétlésbeli hányadik változó első értéke 1, majd 2 stb. miközben a filmek lista első elemére a filmek[0] kifejezéssel hivatkozhatunk? Ezért kell minden lépésben a filmek[hányadik–1] elemet kiírnunk.

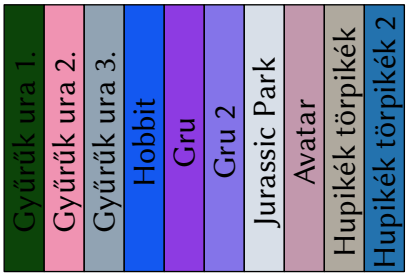
Ugyanezt megtehetjük egy új programszerkezet segítségével, amely könnyebben olvashatóvá teszi a kódot:

```
fut film filmek-ben [
    ki film
]
```

A fut film filmek-ben szerkezet szintén egy ciklust eredményez (hasonlóan az ismétléshez). A kifejezés után szögletes zárójelek közötti utasítások annyiszor hajtódnak végre, ahány eleme van a filmek listának, közben a film változó sorban felveszi a filmek lista elemeinek értékét. A film változónév helyett tetszőleges változónevet használhattunk volna, de könnyíti a kód értelmezését, ha beszédes neveket adunk a változóinknak, ezért volt célszerű a film nevet választanunk.

Ezt az új szerkezetet fogjuk használni a feladatunk megoldására! A DVD-knek csak az oldala látszódik, ezt egy téglalapként fogjuk megjeleníteni, amelynek a felirata lesz a DVD címe. A DVD-borítóknak véletlenszerű színt adunk.

```
balra 90
fut film filmek-ben [
    töltőszín tetszőleges
    téglalap [100, 15]
     szöveg film
    tollatfel hátra 15 tollatle
]
```



A hányadik változó a fut szerkezet esetén is él. Így például könnyen választ kaphatunk arra a kérdésre, hogy hányadik helyen van az Avatar című DVD?

```
fut film filmek-ben [
    ha film == „Avatar” [
        ki hányadik
    ]
]
```

Az elemek indexére vonatkozó kérdésre ciklus nélkül is megkaphatjuk a választ, ha a lista adatszerkezet saját index() eljárását használjuk. Előtte a -ban/-ben segítségével meg kell győződni arról, hogy a keresett elem biztosan megtalálható az adatok között (továbbá mivel a listaelemek indexe 0-tól számozódik, nem pedig 1-től, mint a hányadik, a visszaadott értékhez még hozzáadtunk egyet):

```
ha „Avatar” filmek-ben [
    ki filmek.index(„Avatar”) + 1
]
```

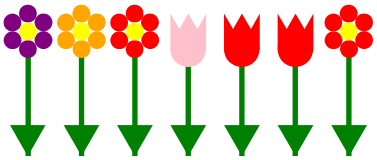
Feladatok

57) Készíts oszlopdiagramot egy listában tárolt értékek alapján!

58) Rajzoljunk ritmusokat lista alapján pl. ritmus = [„ta”, „ta”, „titi”, „ta”, „titi”, „titi”, „ta”, „ta”]

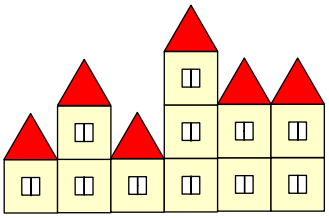
59) Készíts erdő néven eljárást, amely egy lista alapján kirajzolja az erdőt! Például az erdő eljárás fák = [„fa”, „fenyő”, „fa", „fenyő”, „fa”, „fa”, „fenyő”, „fenyő”, „fa”, „fenyő”] listából az alábbi rajzot készíti:

60) Készíts virágoskert néven eljárást, amely egy lista alapján kirajzolja az erdőt. Például a virágok = [[„virág”, „lila”], [„virág”, „narancs”], [„virág”, „piros”], [„tulipán”, „rózsaszín”], [„tulipán”, „piros”], [„tulipán”, „piros”], [„virág”, „piros”]] listából az alábbi rajzot készíti:



Megjegyzés: a példában szereplő lista elemei is listák. Hivatkozhatunk ezek elemeire kettős indexszel is, például a virágok[0][0] értéke „virág”, a virágok[0][1] értéke „lila”. Nincs szükség azonban kettős indexekre, csak egyszeresekre, ha a ciklusváltozó nem a vizsgált elem sorszámát, hanem magát a vizsgált elemet (a kételemű listát) tartalmazza a fut-ban/-ben programszerkezet segítségével.

61) Az építendő lakótelepen lévő házak emeleteinek számát egy lista tartalmazza. Készítsünk a lista alapján a lakótelepről látványtervet! Például, ha a házak = [1, 2, 1, 3, 2, 2] lista tartalmazza a házak emeleteinek számát, akkor a következő látványtervet kapjuk:



Ciklus amíg (feltételes ciklus)

Ezúttal ismét egy ciklusfajtát fogunk megismerni. Mielőtt ezt bemutatnám, térjünk egy kicsit vissza Pisti DVD-s polcára. Legutóbb arra kerestük a választ, hogy hányadik helyen található az Avatar című film. Erre a kérdésre a következő módon kaptunk választ:

```
fut film filmek-ben [
    ha film == „Avatar” [
        ki hányadik
    ]
]
```

Ez a ciklus mindig végigmegy az egész listán (végignézi az egész polcot), miközben ha belegondolunk, elegendő lenne addig vizsgálni a DVD-ket, ameddig meg nem találjuk az Avatar címűt. Ennek főleg akkor van jelentősége, ha rengeteg adatot tartalmazó listával dolgozunk, ugyanis a számítógépnek is szüksége van időre a feladatok megoldásához (még ha néha úgy is tűnik, hogy azonnal oldja meg őket), tehát a programunk írásakor törekednünk kell arra, hogy minél kevesebb lépésben oldja meg a feladatot, ezáltal gyorsabban fog működni.

Tehát azt szeretnénk, hogy csak addig vizsgáljuk a filmeket, ameddig meg nem találtuk az Avatart. Erre szolgál az úgynevezett amíg-os vagy feltételes ciklus (angolul while ciklus). Ez a ciklusfajta addig ismétli a ciklusmagban lévő (LibreLogóban a szögletes zárójelek közötti) utasításokat, amíg egy bizonyos feltétel igaz. Tehát addig kell lépkedni, amíg a vizsgált elem nem egyezik meg az Avatarral. Nézzük meg ennek a megvalósítását:

```
sorszám = 0
amíg filmek[sorszám]!= „Avatar” [
    sorszám = sorszám+1
]
ki sorszám+1
```

Ennél a ciklusfajtánál, ha listaelemeket vizsgálunk, akkor nyilván kell tartanunk, hogy éppen hányadik listaelemnél tartunk, erre szolgál a fenti példában a sorszám nevű változó. Amíg a vizsgált sorszámú film nem az Avatar, addig növeljük a sorszám értékét eggyel. Az ismétlődés akkor áll meg, ha a feltétel már nem teljesül, tehát a soron következő film az Avatar. Ekkor a sorszám a filmek listában az Avatar sorszáma lesz. Mivel a listaelemek számozását 0-tól kezdjük, ezért kell a kiíráskor a sorszámnál eggyel nagyobbat kiírni.

Feltételes ciklusokat leggyakrabban eldöntési, kiválasztási és keresési feladatokban használunk, további példákat találsz a megfelelő programozási tételeknél, a 106. oldaltól kezdődően.

Feladatok

62) Rajzolj ki egyre kisebb négyzeteket: a program kérje be a legnagyobb négyzet oldalhosszát, majd innentől kezdve addig rajzoljon egyre kisebb négyzeteket, amíg az aktuális négyzet oldalhossza nagyobb, mint 1 pont!

63) Készíts tetszőleges alakzatból álló oldalszegélyt úgy, hogy a teknőc akkor forduljon, ha a lap szélétől 1 cm távolságnál közelebb ért, és akkor hagyja abba a mintarajzolást, ha visszaért a kiindulópontba.

Szövegkezelés, szöveg szétvágása

A szövegek a programozás során karakterekből álló láncokként tárolódnak. A karakterláncokat idézőjelek közötti szövegként adhatjuk meg.

k = „”

Ez egy üres karakterláncot jelent.

szó= „példa”

A karakterláncok sok dologban hasonlítanak a listákhoz, például hivatkozhatunk valamelyik sorszámú karakterére, ahogy hivatkozhatunk valamelyik sorszámú listaelemre is.

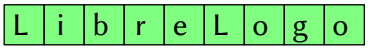
```
ki szó[0]
ki szó[4]
```

A fenti két sor a szó első és ötödik karakterét írja ki. Tehát az is közös a lista és karakterlánc működésében, hogy a sorszámozást 0-tól kezdjük. Ugyanígy kiíratható a szó utolsó karakterét is a listáknál már megismert módon:

```
ki szó[–1]
```

Emellett a fut szerkezetet is használhatjuk a karakterláncokon karakterenként való végiglépkedésre. Így például egy szót kiírhatunk úgy, hogy minden egyes betűje egy különálló négyzet belsejébe kerüljön.

```
szó = „LibreLogo”
fut betű szó-ban [
    négyzet 10
    szöveg betű
    tollatfel jobbra 90 előre 10 balra 90 tollatle
]
```



A karakterláncból részláncokat vághatunk ki, ha a szögletes zárójelben megadjuk, hogy a karakterlánc hányadik karakterétől hányadik karakteréig lévő részláncra van szükségünk.

```
szó = „LibreLogo”
ki szó[0:5] ; a szó[5] karaktert megelőző részlánc: „Libre”
ki szó[2:5] ; a szó[2]-től a szó[5] karakter előttig lévőrészlánc: „bre”
ki szó[5:]  ; a szó[5] karaktertől lévő részlánc: „Logo”
ki szó[:–1] ; az utolsó karaktert megelőző részlánc: „LibreLog”
 ki szó   ; az eredeti szó: „Librelogo”
```

Megjegyzés: *Ez a művelet listák esetén is létezik, akkor eredményül részlistákat kapunk.*

A részláncok segítségével elkészíthetjük például ezt a piramist:

```
szó = „LibreLogo”
amíg szó != „” [
    címke szó
    szó = szó[:-1]
    tollatfel előre 10 tollatle
]
```

L
Li
Lib
Libr
Libre
LibreL
LibreLo
LibreLog
LibreLogo

A karakterláncokat egymás után fűzhetjük a + művelet segítségével. Erre akkor lehet szükség, ha több láncot szeretnénk egy szövegdobozba, vagy alakzat belsejébe írni, vagy a ki utasítással kiírni a képernyőre.

```
szó1 = „Libre”
szó2 = „Logo”
ki szó1+szó2
szó = szó1+szó2
címke szó
```

LibreLogo

Számokat is átalakíthatunk karakterlánccá a lánc utasítást használva, így például számjegyenként dolgozhatunk a számmal. A következő példában ez a szám 192*643, azaz 123 456:

```
szám = lánc 192 * 643
fut számjegy szám-ban [
    négyzet 15
    szöveg számjegy
    tollatfel jobbra 90 előre 15 balra 90 tollatle
]
```

1	2	3	4	5	6

A karakterlánc hosszát, ahogy a listák elemszámát is, a darab utasítás segítségével kaphatjuk meg:

címke **darab „példa”**

5

Ha hosszabb karakterláncot helyezünk egy szövegdobozba, akkor előfordul, hogy több sorba szeretnénk a szöveget írni. Szerencsére nem kell minden sor számára külön szövegdobozt készíteni, hanem elegendő sortörés karaktert írnunk a karakterlánc megfelelő helyére. Ennek jele \n:

címke „Első sor.**\n**Második sor.”

Első sor.
Második sor.

A karakterláncok tartalmazhatnak szóközöket is:

mondat = „Tetszőleges mondat lehet karakterlánc. Sőt ez és az előző mondat együtt is alkothat karakterláncot.”

A hosszabb karakterláncokkal való munka során gyakran előfordulhat, hogy azokat szét kell szednünk rövidebb karakterláncokká. A LibreLogóban erre van egy függvény, amely tetszőleges karakternél szétvágja a karakterláncokat több részlánccá, amelyeket egy listában ad meg. Így a fenti mondat nevű karakterlánc szétvágható a „.” karakternél két karakterláncból álló listává a split()[7] függvény segítségével:

mondatlista = **mondat.split(„.”)**
ki mondatlista ; kimenet: [„Tetszőleges mondat lehet karakterlánc”, „ Sőt ez és az előző mondat együtt is alkothat karakterláncot”]

A split() függvény a karakterlánc típus saját függvénye, ezért kapcsoljuk egy ponttal a karakterlánchoz, amelyre alkalmazni szeretnénk. Zárójelek között adjuk meg a „töréspontot”, vagyis azt a karakterlánc-részletet, amelynek előfordulási helyein több részláncra vágja az adott szöveget a split függvény. Ha a split nem talál egy töréspontot sem a szövegben, akkor az eredeti szöveget kapjuk vissza egy egyelemű listában. Ha nem adunk meg paramétert a split-nek, akkor alapértelmezés szerint a szóközöknél választja szét a szöveget:

szavaklista = **mondat.split()**
ki szavaklista ; kimenet: [„Tetszőleges”, „mondat”, „lehet”, „karakterlánc.”, „Sőt”, „ez”, „és”, „az”, „előző”, „mondat”, „együtt”, „is”, „alkothat”, „karakterláncot.”]

[7] A „split”, magyarul „széthasít”, valójában egy Python függvény. A LibreLogo ebben a programozási nyelvben íródott, és a legtöbb Python utasítás elérhető a LibreLogóban is.

A split függvény mellett további hasznos függvények állnak rendelkezésünkre. Például nagybetűssé vagy kisbetűssé, vagy akár nagy kezdőbetűssé alakíthatjuk a szövegünket:

```
szavak = „ez egy tetszőleges KARAKTERLÁNC”
címke szavak.upper()
```

EZ EGY TETSZŐLEGES KARAKTERLÁNC

```
címke szavak.lower()
```

ez egy tetszőleges karakterlánc

```
címke szavak.capitalize()
```

Ez egy tetszőleges KARAKTERLÁNC

A szétvágás, a nagy kezdőbetűsítés és az összefűzés műveletének segítségével a fenti szavak nevű karakterlánc minden szavát nagy kezdőbetűssé alakíthatjuk a következő programmal:

```
szavak = „ez egy tetszőleges KARAKTERLÁNC”
szavak = szavak.lower()
szavak = szavak.split()
újszavak = „”
fut szó szavak-ban [
    újszavak = újszavak + szó.capitalize() + „ ”
]
szavak = újszavak[:–1]
címke szavak
```

Ez Egy Tetszőleges Karakterlánc

Először a szavak karakterláncot csupa kisbetűssé alakítjuk, majd szétvágjuk a szóközök mentén, a kapott listát eltároljuk a szavak változóban. Létrehozunk egy újszavak nevű üres karakterláncot, ehhez fogjuk hozzáfűzni a szavak lista egyenként nagybetűssé alakított szavait. Minden szó után egy szóközt is hozzá kell fűznünk a szöveghez, emiatt az újszavak karakterlánc végére kerül egy felesleges szóköz, azt a részlisták műveletével levágjuk, majd a kapott új karakterláncot visszatesszük a szavak változóba.[8]

[8] Számos rövidítésre van lehetőség. A típusműveletek (metódusok) sorozata például egy programsorba is összevonható: szavak = szavak.lower().split(). Sőt, a title() karakterlánc-metódus pont azt végzi el, amit a programunk nagy része, a karakterlánc szavainak nagykezdőbetűsítését. Így a teljes programunk átírható egy programsorba: címke „ez egy tetszőleges KARAKTERLÁNC”.lower().title()

Ahogy azt már láttuk, a karakterláncok kezelése sok mindenben megegyezik a listákéval. A rendez függvény sem kivétel, karakterlánc esetében visszaadja a karakterlánc karaktereinek rendezett listáját.

Feladat: Készítsünk programot, amely eldönti, hogy két szó egymás anagrammája-e (betűik sorrendjét megváltoztatva megkaphatjuk-e az egyiket a másikból)!

Ez a feladat a 2010. októberi érettségi programozási feladatának egy részfeladata volt. A feladat kiválóan megoldható rendezett listák segítségével. A megoldás elve a következő. Mindkét szó (azaz karakterlánc) betűit rendezett listává alakítjuk a rendez függvénnyel. A két rendezett lista pontosan akkor fog megegyezni, ha a két szó egymás anagrammája volt.

```
szó1 = be „Add meg az első szót!”
szó2 = be „Add meg a második szót!”
ha szó1 != szó2 és (rendez szó1) == (rendez szó2) [
    ki „Anagrammák.”
][
    ki „Nem anagrammák.”
]
```

A karakterláncokkal kapcsolatos műveleteknek majd a szöveges fájlokkal való munka során vesszük igazán nagy hasznát, erről részletesebben az utolsó két fejezetben olvashatsz.

Feladatok

64) Készíts a „VAKÁCIÓ” szövegből piramist!

Ó
IÓ
CIÓ
ÁCIÓ
KÁCIÓ
AKÁCIÓ
VAKÁCIÓ

65) Olvass be egy szöveget, majd írd ki szavanként különböző színnel! A feladatban segítséget jelent a címke2 utasítás, amellyel a teknőc az előző címke2 utasítással kiírt szöveg után írja ki az új szöveget.

Színezd a szöveget szavanként különböző színűre!

66) Jelenítsd meg Tóth Ágnes *Lila vers* című költeményét úgy, hogy a színeket jelentő szavak (viola, ibolya, lila) a színnek megfelelő betűszínűek legyenek!

Harmatcseppben fürdik
A szerény ibolya
Mellette öltözik
Az esti viola
Lila szoknya libben
A kökörcsin lányon
Abban fog táncolni
A pillangó bálon
Szivárvány hídján
Lila szín várja
Legyen a leglilább
Virág a párja

Szabályos kifejezések, keres, talál, cserél függvények

A szabályos, más néven reguláris kifejezésekkel megvalósított mintaillesztéssel összetett karakterlánc-műveleteket írhatunk le rendkívül tömören. Például a (később részletesen megmagyarázott) \b[abc]\w* szabályos kifejezés *a*, *b* vagy *c* betűvel kezdődő, tetszőlegesen hosszú szóra illeszkedik egy szövegben. Az így kiválasztott szót, vagy szavakat egy lépésben kinyerhetjük, átalakíthatjuk a megfelelő függvény segítségével, ráadásul legtöbbször jóval gyorsabban, mintha egyszerű karakterlánc-kezelő függvényekkel dolgoznánk.

A szabályos kifejezéseket mára a legtöbb programozási nyelv és néhány irodai program is ismeri (például a LibreOffice *Keresés és csere* menüpontjában is használhatunk szabályos kifejezéseket). Elsőre nehéznek tűnhetnek, de sok példát fogok mutatni, amelyek alapján már érthető lesz a használatuk.

A következő táblázat a legalapvetőbb szabályos kifejezéseket tartalmazza. Ezeket majdnem tetszőlegesen lehet kombinálni egymással, így a szabályos kifejezések tárháza szinte végtelen.

Kifejezés	Jelentése
Kati	A „Kati” karakterláncra illeszkedő kifejezés.
.	Tetszőleges karakterre illeszkedik.
Kat.	A „Kat”-ra és egy tetszőleges karakterre illeszkedik, pl. ez lehet a *Kata*, *Kati*, *Katt, Katq* stb. karakterlánc a szövegben.
?	Az előtte lévő karakter 0 vagy 1 előfordulására illeszkedik.
Katt?i	A „Katti” és a „Kati” szóra is illeszkedő kifejezés.
*	Az előtte lévő karakter 0 vagy többszöri ismétlődéséből álló láncra illeszkedik. *Megjegyzés: ez az illeszkedés „mohó”, azaz minél nagyobb karakterláncra illeszkedik a szabályos kifejezés. Ha ennek az ellenkezőjét szeretnénk, azaz a minél rövidebb illeszkedést (pl. egy összetett szabályos kifejezésben), tegyünk egy kérdőjelet a csillag (vagy az előbb szereplő kérdőjel, vagy a következőkben ismertetett pluszjel után): *?, ??, +?*
.*	Tetszőleges karakter 0 vagy többszöri ismétlődéséből álló láncra illeszkedik.
+	Az előtte lévő karakter 1 vagy többszöri ismétlődéséből álló láncra illeszkedik.

Kifejezés	Jelentése
.+	Tetszőleges karakter 1 vagy többszöri ismétlődéséből álló láncra illeszkedik.
[betűk]	Egy karakterre illeszkedik a „b”, „e”, „t”, „ű”, „k” karakterek közül.
[a-d]	A latin ábécé a-tól d-ig terjedő tartományából (azaz az „a”, „b”, „c”, „d” betűk közül) egy karakterre illeszkedik.
[a-z]	A latin ábécé egy tetszőleges kisbetűs karakterére illeszkedik.
[A-Z]	A latin ábécé egy tetszőleges nagybetűs karakterére illeszkedik.
[0-9]	Egy tetszőleges számjegyre illeszkedik.
[a-Z0-9]	Egy latin betűre vagy számjegyre illeszkedik.
^	A karakterlánc elejére illeszkedik.
$	A karakterlánc végére illeszkedik.
\b	Szó elejére vagy végére illeszkedik.
^a	Egy „a”-val kezdődő karakterláncra illeszkedik.
a$	Egy „a”-ra végződő karakterláncra illeszkedik.
\w	Tetszőleges betűre (ékezetesekre is), számra vagy aláhúzásjelre illeszkedik.
[.]	A „.” karakterre illeszkedik.
[-.*+^]	A „-”, „.”, „*”, „+”, „^” karakterek bármelyikére illeszkedik.[9]
(Kati)	A „Kati” szóra illeszkedik. (A szabályos kifejezések zárójeles részkifejezéseire a későbbiekben vissza tudunk hivatkozni.)
(Kati|Peti)	A „Kati” vagy a „Peti” szóra illeszkedő kifejezés.
(Ka|Pe)ti	A „Kati” vagy a „Peti” szóra illeszkedő kifejezés.
{*n*}	Az előtte lévő karakter vagy zárójeles részkifejezés *n*-szeri ismétlődéséből álló láncra illeszkedik. Pl. az a{3} kifejezés az „aaa” karakterlánc-részletre illeszkedik.
n	Visszahivatkozás az *n*-dik részkifejezésre illeszkedő karakterláncra, így az illeszkedés csak adott részkifejezés ismétlődése esetén teljesül. Például a (Kat.)\\1 a *KataKata*, *KatiKati* stb. karakterláncra illeszkedik, a *KataKati*-ra nem.

[9] A kötőjel helye itt kötött: csak elöl szerepelhet (különben karaktertartományt jelöl).

Most már értelmezhetjük a bevezető szakaszban látott \b[abc]\w* kifejezést: illeszkedés a szó eleji (\b) lévő *a*, *b* vagy *c* betűkre ([abc]), és az azt követő, legalább egy betűből álló betűsorozatra (\w+).

A LibreLogóban a keres(), a talál(), a cserél() függvények használatához van szükség a szabályos kifejezésekre.

A keres() függvénynek két paramétere van, az első paramétere egy szabályos kifejezés, a második pedig egy karakterlánc, a függvény eredménye egy logikai érték, azaz igaz, ha a karakterlánc tartalmaz a szabályos kifejezésnek megfelelő szövegrészt, különben pedig hamis. A keres() függvényt tehát elágazás, vagy amíg-os ciklus feltételeként használhatjuk.

```
szó = be „Írj be egy szót!”
ha keres(„a”, szó) [
    ki „Van benne a.”
][
    ki „Nincs benne a.”
]
```

A fenti példa először beolvas egy szót, majd a keres() függvény és egy elágazás segítségével eldönti, hogy tartalmaz-e a beolvasott szó „a” karaktert.[10]

Ezt használhatjuk a beolvasott értékek ellenőrzésére. Például ha számértéket várunk a beolvasáskor, akkor csak számjegyeket fogadjunk el, és amíg nem számjegyeket ad meg a felhasználó, addig újra és újra kérjük tőle az értéket. Ennek megvalósítása a következő módon lehetséges:

```
szám = be „Adj meg egy számot!”
amíg nem keres(„^[0-9]+$”, szám)[
    ki „Nem számot adtál meg!”
    szám = be „Adj meg egy számot!”
]
szám = egész szám
```

Először beolvastuk a számot a szám változóba, majd amíg nem teljesül, hogy a beolvasott érték számjegyekből áll, addig írjuk ki, hogy *„Nem számot adtál meg!”* és újra beolvassuk a számot, csak ezután alakítjuk át a számot egésszé. Itt a „^[0-9]+$” szabályos kifejezésből a „[0-9]+” fejezi ki, hogy egy vagy több számjegyet várunk, a „^”, és „$” pedig, hogy ezt a számsorozatot a szöveg két vége kell, hogy határolja, tehát számjegyeken kívül más már nem lehet a szövegben.

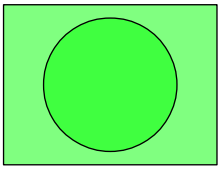
[10] Ezt az egyszerű feltételt írhatjuk persze így is: ha „a” szó-ban.

A talál() függvény paraméterei megegyeznek a keres() függvény paramétereivel, azaz egy szabályos kifejezést és egy karakterláncot vár paraméterként, azonban eredménye egy lista, amely a paraméterként megadott szöveg azon részláncait tartalmazza, amelyek megfelelnek az adott szabályos kifejezésnek. Például egy szövegrészletből kinyerhetjük a számokat a következő módon:

```
mondat = „A téglalap egy oldala 80, a másik oldala 60 pont legyen. A téglalapba rajzolj egy 50 pont átmérőjű kört!”
számok = talál(„[0-9]+”, mondat)
ki számok
```

A talál() függvény kigyűjtötte a mondat karakterláncból a legalább egy számjegyet tartalmazó szövegrészeket, eredményét a számok változóban tároltuk, amelynek így az értéke a [„80”, „60”, „50”] lista lett. A lista elemei természetesen karakterláncok, tehát ha számolni szeretnénk velük, akkor egyenként számmá kell alakítanunk őket.

```
mondat = „A téglalap egy oldala 80, a másik oldala 60 pont legyen. A téglalapba rajzolj egy 50 pont átmérőjű kört!”
számok = talál(„[0-9]+”, mondat)
újszámok =[]
fut szám számok-ban [
    újszámok = újszámok + [egész szám]
]
számok = újszámok
téglalap [számok[0], számok[1]]
kör számok[2]
```



A lista elemeinek számmá alakításához létre kell hoznunk egy segédváltozót, a példában erre szolgál az újszámok nevű változó, amely kezdetben egy üres lista, majd a fut szerkezet minden lépésében hozzáfűzzük a számok lista következő elemét számmá alakítva. Ezután az újszámok értékét visszatesszük a számok változóba. Így a számok lista elemeit már használhatjuk a téglalap és a kör eljárások paramétereiként.

A cserél() legalább három paramétert vár: cserél(mit, mire, miben). A „mit” egy szabályos kifejezés, amely meghatározza, hogy a „miben” karakterlánc mely rész-

láncait cseréljük ki a „mire” karakterláncra. A függvény eredménye a cserék utáni karakterlánc lesz.

```
mondat = „Kiszedhetjük a mondatból a magánhangzókat.”
címke cserél(„[aáeéiíoóöőuúüű]”, „_”, mondat)
```

K_sz_dh_tj_k _ m_nd_tb_l _ m_g_nh_ngz_k_t.

A fenti példában a mondat változóban tárolt karakterláncban a magánhangzókat a „_” karakterre cseréltük a cserél() függvénnyel, az eredményeként kapott karakterláncot pedig egy szövegdobozba írtuk.

A cserél() a szabályos kifejezésre illeszkedő összes szövegrészletet cseréli. Ha viszont az első meghatározott számú illeszkedést szeretnénk cserélni, akkor ezt megtehetjük egy negyedik paraméter megadásával:

```
mondat = „Kiszedhetjük a mondatból a magánhangzókat.”
címke cserél(„[aáeéiíoóöőuúüű]”, „_”, mondat, 10)
```

K_sz_dh_tj_k _ m_nd_tb_l _ m_ganhangzókat.

Lehetőség van a „mire” karakterláncban az illeszkedő mintát is ismételten beilleszteni a \g<0> kifejezéssel. Az illeszkedő minta zárójeles részkifejezéssel meghatározott részeit úgyszintén, a szabályos kifejezéseknél már megismert *n* mintával. Például egy szöveget dadogóssá tehetünk úgy, hogy a szavakban a mássalhangzóval kezdődő szótagok elejét megismételjük:

```
mondat = „Dadogós mondat.”
címke cserél(„[BCDFGHJKLMNPQRSTXVZbcdfghjklmnpqrstvz][aáeéiíoóöőuúüű]”,
    „\g<0>-\g<0>-”, mondat)
```

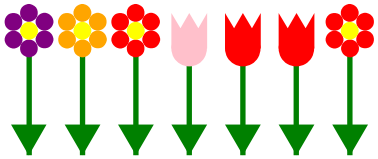
Da-Da-do-do-gó-gó-s mo-mo-nda-da-t.

Feladatok

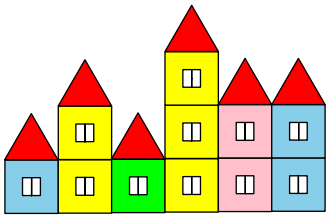
67) Készíts órát rajzoló eljárást, amely egy „óra:perc” alakú szöveg alapján rajzolja ki az óramutatókat!

68) Készíts eljárást, amely karakterlánc alapján kétféle fákból álló erdőt rajzol! Például a „fa fe fa fe fa fa fe fe fa fe” szövegből az alábbi ábrát készíti:

69) Készíts eljárást, amely karakterlánc alapján kétféle virágból álló kertet rajzol! Pl. „v-lila v-narancs v-piros t-rózsaszín t-piros t-piros v-piros” szövegből az alábbi ábrát készíti:



70) Készíts eljárást, amely karakterlánc alapján emeletes házakból álló sorházat rajzol! Pl. „1-világoskék 2-sárga 1-világoszöld 2-sárga 2-rózsaszín 2-világoskék” szövegből az alábbi ábrát készíti:



71) Készíts eljárást a bekért szöveg „madárnyelvvé” alakítására! Például a *„Te tudsz így beszélni?”* szöveg esetén a kimenet legyen *„Teve tuvudsz ívígy beveszévélnivi?”*

Halmazok

Ezúttal ismét egy összetett adatszerkezettel fogunk megismerkedni, amely nem más, mint a *halmaz*. A halmaz egy olyan több elemből álló adattípus, amelyben az elemeknek nincs sorrendje, és minden elem pontosan egyszer szerepel a halmazban, azaz az elemek nem ismétlődhetnek. A LibreLogo halmaz típusa a listákhoz hasonlóan a Python nyelv megfelelő adatszerkezetének felel meg. Elemei különböző típusú elemek is lehetnek, azaz egy halmaznak lehet eleme szám, szöveg, logikai érték, de akár lista vagy halmaz is. A halmazokat az elemek { } közötti felsorolásával jelöljük. Azaz, az alábbi sor egy halmazt ír le, amelynek eleme egy szöveg, egy szám és egy lista.

```
{„alma”, 2, [1, 2, 3]}
```

A fenti halmaz megegyezik az alábbi halmazzal, hiszen a halmaznál az elemeknek nincs sorrendje:

```
{2, „alma”, [1, 2, 3]}
```

Tetszőleges listát átalakíthatunk halmazzá a halmaz utasítás segítségével, amelynek paramétere a lista, eredménye pedig a kapott halmaz:

```
K = halmaz [1, 2, 2, 3, 3, 4, 5]
ki K
```

A K értéke a {1, 2, 3, 4, 5} halmaz lett, tehát a lista ismétlődő elemeit elhagytuk a halmazzá alakítás során. Mivel a halmaz elemeinek nincs sorrendje, ezért a halmaz elemeire nem hivatkozhatunk az indexükkel. Ha ezzel próbálkozunk, akkor hibaüzenetet kapunk:

```
K = halmaz [1, 2, 2, 3, 3, 4, 5]
ki K[0]
```

„ Hiba (hiányzó vagy felesleges szóköz a kapcsos zárójelnél?) ”

A halmazok esetén tehát nem az a fontos, hogy egy eleme hányadik helyen áll, hanem csak az, hogy benne van-e a halmazban, vagy sem. Ezt a listáknál és a karakterláncoknál már látott -ban/-ben kifejezéssel vizsgálhatjuk meg, eredményül egy logikai értéket, igaz vagy hamis- t kapva:

```
páros = halmaz sor 0 20 2
szám = egész(be „Adj meg egy 20-nál kisebb pozitív egész számot!”)
ha szám páros-ban [
    ki „A szám páros.”
][
```

```
    ki „A szám páratlan.”
]
```

A fenti példában a páros halmaz elemeit a listáknál bemutatott sor utasítással adtuk meg (0-tól kettesével haladva 20-ig lévő számokból álló halmazt készítettünk). Majd beolvastunk egy 20-nál kisebb pozitív egész számot, amelyről eldöntöttük, hogy eleme-e a páros halmaznak, ha igen, akkor kiírtuk, hogy páros, különben pedig azt, hogy páratlan a szám.

A halmaz *elemszámát* a listákhoz hasonlóan megkaphatjuk a darab utasítás segítségével:

```
K = halmaz [1, 2, 3, 4, 2, 3, 5]
ki darab K
```

Gyakori kérdés lehet két halmazzal kapcsolatban, hogy adjuk meg a közös elemeiket. A két halmaz közös elemeiből álló halmazt a *két halmaz metszetének* nevezzük. Két halmaz metszetét megkaphatjuk az &[11] műveleti jel segítségével:

```
C = {1, 2, 3, 4, 5}
D = {2, 4, 6, 8, 10}
ki C&D
```

A fenti példában a {2, 4} halmaz kerül kiírásra.

Egy másik gyakori halmazművelet, amikor azokat az elemeket keressük, amely a két halmaz valamelyikében benne vannak. Ezekből az elemekből álló halmazt nevezzük a *két halmaz uniójának*. Két halmaz unióját a | műveleti jellel állíthatjuk elő:

```
C = {1, 2, 3, 4, 5}
D = {2, 4, 6, 8, 10}
ki C|D
```

A két halmaz uniója az előző példában a {1, 2, 3, 4, 5, 6, 8, 10} halmaz.

A harmadik halmazművelet *két halmaz különbsége*, itt azokat az elemeket keressük, amelyek az egyik halmaznak az elemei, de a másiknak nem. Ez a művelet az unió és a metszettel ellentétben nem egy szimmetrikus művelet, azaz nem mindegy, hogy melyik halmazból melyiket vonjuk ki. A halmazok különbségét a – jel segítségével adjuk meg:

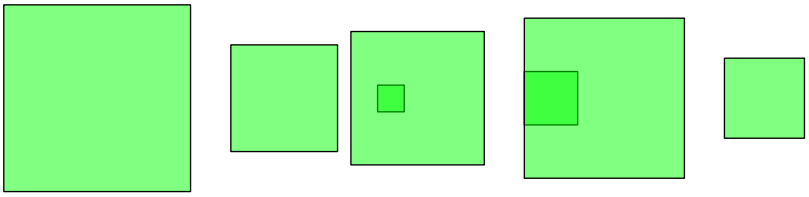
```
C = {1, 2, 3, 4, 5}
D = {2, 4, 6, 8, 10}
ki C-D
ki D-C
```

[11] Ejtsd: „et jel” vagy „és jel”.

A C-D = {1, 3, 5} halmaz, a D-C = {6, 8, 10} halmaz lesz.

Bár a halmazok elemeit nem indexelhetjük, a fut -ban/-ben szerkezet segítségével végiglépkedhetünk a halmaz elemein:

```
A = {10, 20, 30, 40, 50, 60, 70}
fut elem A-ban [
    négyzet elem
    tollatfel jobbra 90 előre elem balra 90 tollatle
]
```



A példából látszik, hogy a végiglépkedés sorrendje tetszőleges, nem egyezik meg azzal a sorrenddel, amelyben az elemeket megadtuk.

Feladatok

72) Készíts programot, amely két szóról eldönti, hogy ugyanazokból a karakterekből épülnek-e fel!

73) Egy baráti társaság beszélget, hogy mely országokban jártak eddig. A válaszaik alapján határozd meg, hogy melyek azok az országok, amelyekben járt közülük legalább egyvalaki! A barátok számát és a válaszaikat a felhasználó adja meg.

74) Egy baráti társaság beszélget, hogy mely országokban jártak eddig. A válaszaik alapján határozd meg, hogy mely országokban járt közülük mindenki!

75) Lili születésnapjára színes ceruzákat kapott ajándékba. Az ajándékozók nem beszéltek össze, ezért néhány színből többet is kapott. Olvasd be, hogy milyen színű ceruzákat kapott Lili, és minden színből rajzolj ki egy ceruzát a képernyőre!

Programozási tételek megvalósítása LibreLogóban

Ebben a fejezetben megismerkedünk néhány alapvető programozási problémával, amelyre egy hatékony és garantáltan helyes megvalósítást is adunk. Ezeket a mintamegoldásokat nevezzük programozási tételeknek. Egy programozási tétel adott típusú problémákra ad megoldást.

A továbbiakban bemutatom, hogy az egyes tételeket milyen feladatok megoldása során használjuk, majd a felsorolt példákból néhányat megbeszélünk, a többit meghagyom az olvasónak önálló feldolgozásra.

Eldöntés

Az *eldöntés* programozási tételt olyan feladatokban használjuk, amelyeknél arra vagyunk kíváncsiak, hogy van-e bizonyos tulajdonságú elem egy sorozatban (listában). Ezt a programozási tételt használjuk például az alábbi feladatok megoldása során:

- Van-e a virágoskertben tulipán?
- Van-e az osztályban 160 cm-nél magasabb diák?
- Van-e a jelenlevők között V betűvel kezdődő nevű?
- Minden jelenlevő neve kevesebb, mint 10 betűből áll?
- Van-e két ember, akiknek a neve ugyanazzal a betűvel kezdődik?
- Van-e az elemek között négyzetszám?
- Döntsük el egy egész számról, hogy prímszám-e!

A fenti kérdések elég általánosak, ezért a példákat tovább pontosítom. A most részletesen be nem mutatott példákhoz a fejezet végén találsz pontos leírást, amely alapján megoldhatod a feladatokat.

Feladat: Egy parkban különféle virágágyások vannak egymás mellett, egy virágágyás azonos virágokat tartalmaz. A virágágyásokról nyilvántartást vezetnek, hogy melyikben milyen virágot ültettek. A nyilvántartás alapján döntsük el, hogy van-e a parkban tulipán!

Ha a nyilvántartásunk egy lista, amely a virágágyások fajtáját tartalmazza, akkor pillanatok alatt válaszolhatunk a kérdésre:

```
nyilvántartás = [„jácint”, „árvácska”, „tulipán”, „százszorszép”, „margaréta”]
ha „tulipán” nyilvántartás-ban [
    ki „Van tulipán.”
][
```

```
    ki „Nincs tulipán.”
]
```

A megoldás azért volt ilyen egyszerű, mert elegendő volt azt vizsgálnunk, hogy egy konkrét elem benne van-e a nyilvántartás listában. Lássunk hát egy kicsit nehezebb feladatot.

Feladat: Egy osztály orvosi vizsgálaton vett részt, amelyen a testmagasságukat is megmérték. Döntsd el a mérési eredményeket tartalmazó lista alapján, hogy van-e az osztályban 160 cm-nél magasabb diák? (A példában véletlenszerű értékeket fogok vizsgálni.)

```
n = egész véletlen 40  ; az osztály létszáma egy 40-nél kisebb egész szám
magasságok = [] ; létrehozom az üres magasságok listát
ismétlés n [
    ; hozzáadom a következő véletlenszerű magasságot:
    magasságok = magasságok + [100 + véletlen 70]
]
ki magasságok
```

A fenti kóddal előállítottam egy legfeljebb 39 fős osztályt, 100 és 170 cm közé eső magassággal. Most jön az eldöntés tétel megvalósítása, amelynek alapötlete a következő: addig lépkedek a lista elemein, amíg a soron következő elem nem felel meg a feltételemnek, vagy a lista végére nem érek. Ha előbb álltam meg, minthogy elérjem a lista végét, akkor volt a feltételnek megfelelő elem a listában, különben pedig nem. A feltételig való ismétlést természetesen amíg-os ciklussal valósíthatjuk meg.

```
i = 0
amíg (i < darab magasságok) és (magasságok[i] <= 160) [
    i = i+1 [12]
]
ha i < darab magasságok [
    ki „Volt 160 cm-nél magasabb.”
][
    ki „Nem volt 160 cm-nél magasabb.”
]
```

Hasonlóan oldható meg a következő feladat is.

Feladat: Egy rendezvény résztvevőinek neveit a nevek listában tároljuk. Készítsünk programot, amely eldönti, van-e a jelenlevők között „V” betűvel kezdődő nevű?

[12] Használható **i += 1** alakban is.

A neveket egyesével olvassuk be a nevek listába:

```
nevek = []
név = be „Add meg a következő résztvevő nevét!”
amíg név != „” [
    nevek = nevek + [név]
    név = be „Add meg a következő résztvevő nevét!”
]
```

A beolvasott listán most is egy feltételes ciklussal lépkedünk végig, amíg a végére nem érünk, vagy amíg „V”-vel kezdődőt nem találunk (azaz a karakterlánc 0. karaktere nem „V”). Ha előbb álltunk meg, minthogy a lista végére értünk volna, akkor volt „V”-vel kezdődő nevű, különben pedig nem.

```
i = 0
amíg (i < darab nevek) és (nevek[i][0] != „V”) [
    i = i+1
]
ha i < darab nevek [
    ki „Volt V-vel kezdődő.”
][
    ki „Nem volt V-vel kezdődő.”
]
```

Az *Eldöntés* tétel másik fontos alkalmazása a következő példához hasonló feladatok, amikor arra vagyunk kíváncsiak, hogy a listában minden elem rendelkezik-e egy bizonyos tulajdonsággal. Az ilyen feladatoknál is feltételes ciklust használunk. Addig lépkedünk a ciklusban tovább, ameddig az utolsó elemhez nem érünk, vagy az elem megfelel a tulajdonságnak. Ha a lista végére értünk, akkor minden elem rendelkezett a tulajdonsággal, különben pedig volt olyan elem, amely nem az adott tulajdonságú.

Feladat: Egy rendezvényen minden résztvevő névre szóló oklevelet kap. Kérdés, hogy minden résztvevő neve kifér-e egy 10 karakter széles helyre. Készítsünk programot, amely a nevek ismeretében válaszol erre a kérdésre. (A neveket az előző feladatban használt módszerrel olvassuk be.)

```
nevek = []
név = be „Add meg a következő résztvevő nevét!”
amíg név != „” [
    nevek = nevek + [név]
    név = be „Add meg a következő résztvevő nevét!”
]
i = 0
```

```
amíg (i < darab nevek) és (darab(nevek[i]) <= 10) [
    i = i+1
]
ha i == darab nevek [
    ki „Minden résztvevő neve kifér a 10 karakter széles helyen.”
][
    ki „Van olyan, akinek neve nem fér ki a 10 karakter széles helyen.”
]
```

Kiválasztás

A *kiválasztás* tételt olyan feladatokban használjuk, amikor tudjuk, hogy a keresett elem szerepel a listában, és arra vagyunk kíváncsiak, hogy a lista hányadik helyén szerepel ez az elem, vagy mi a pontos értéke.

- Adjuk meg a legkorábbi dátumot, amelyen „A” betűvel kezdődő név névnapja van!
- Adjunk meg két azonos hosszúságú nevet!
- Adjuk meg egy 2-nél nagyobb egész szám legkisebb 1-től különböző osztóját!
- Adjuk meg a legkisebb n-jegyű négyzetszámot!
- Adjuk meg a legkisebb n-jegyű prímszámot!

A névnapos kérdéseket majd a fájlokkal való munka megismerése után fogom megmutatni. Ezért most a harmadik feladat bemutatásával kezdem.

Feladat: Olvassunk be egy 2-nél nagyobb egész számot, majd adjuk meg a legkisebb 1-től különböző osztóját!

```
szám = egész(be „Adj meg egy 2-nél nagyobb egész számot!”)
osztó = 2
amíg szám % osztó != 0 [
    osztó = osztó + 1
]
ki osztó
```

Az első lehetséges osztó a 2. Minden lehetséges osztóra megvizsgáljuk, hogy osztója-e a beolvasott számnak (azaz az osztási maradék 0). Ha nem, akkor eggyel növeljük az osztó értékét. Ahol megáll a ciklus, az lesz a legkisebb osztója a számnak. A ciklus biztosan meg fog állni, mert minden 2-nél nagyobb számnak van 2-nél nagyobb osztója, ha más nem, akkor saját maga.

A kiválasztás tétel annyiban különbözik az eldöntés tételtől, hogy elegendő az amíg-os ciklusban egyetlen megállási feltételt megadnunk, mert biztosan tudjuk, hogy van olyan érték, amilyet keresünk.

Nézzük a következő példát!

Feladat: Adjuk meg a legkisebb *n*-jegyű négyzetszámot!

```
n = egész(be „Adj meg egy 10-nél kisebb pozitív egész számot!”)
szám = 1
amíg szám**2 < 10**(n–1) [
    szám = szám + 1
]
ki szám**2
```

Ehhez tudni kell, hogy a legkisebb *n*-jegyű számot a 10^{n-1} képlettel kapjuk meg. Addig vizsgáljuk az egész számok négyzeteit, ameddig el nem érjük a legkisebb *n*-jegyű számot. Az első ilyen négyzetszám lesz a legkisebb *n*-jegyű négyzetszám.

Keresés

A *keresés* tétel az *eldöntés* és a *kiválasztás* tétel kombinációja. Olyan kérdések megválaszolására használjuk, amikor nem tudjuk, hogy van-e adott tulajdonságú elem a listában. Ha van, akkor megadjuk egy ilyennek a sorszámát vagy az értékét, különben pedig azt mondjuk, hogy nincs ilyen elem a listában.

- Van-e a virágoskertben tulipán, ha igen, akkor milyen színű?
- Van-e az osztályban 160 cm-nél magasabb diák, ha igen, akkor ki az?
- Van-e a jelenlevők között „V” betűvel kezdődő nevű, ha igen, akkor hogy hívják?
- Van olyan jelenlevő, akinek neve több, mint 10 betűből áll, ha igen, akkor ki az?
- Van-e két ember, akiknek a neve ugyanazzal a betűvel kezdődik, ha igen, kik ők?
- Van-e az elemek között négyzetszám, ha van, melyik szám az?
- Van-e az elemek között prímszám, ha van, melyik szám az?

Mivel a *keresés* tétel az e*ldöntés* tétel kiegészítése (azzal, hogy ki is kell választanunk a feltételnek megfelelő elemeket), az *eldöntés* tételnél már bemutatott feladatokat fogom bemutatni egy kis módosítással.

Feladat: Egy parkban különféle virágágyások vannak egymás mellett. Egy virágágyás azonos virágokat tartalmaz. A virágágyásokról nyilvántartást vezetnek, hogy melyikben milyen virágot ültettek. A nyilvántartás alapján döntsük el, hogy van-e a parkban tulipán, és ha van, akkor adjuk meg, hányadik virágágyásban!

A nyilvántartásunk ismét egy lista, amely a virágágyások fajtáját tartalmazza. A megoldás ennél a feladatnál azzal bonyolódik, hogy meg kell adnunk a tulipá-

nos virágágyás sorszámát, emiatt ennél a feladatnál is feltételes ciklust kell használnunk, nem elegendő a -ban/-ben művelet.

```
nyilvántartás = [„jácint”, „árvácska”, „tulipán”, „százszorszép”, „margaréta”]
i = 0
amíg (i < darab nyilvántartás) és (nyilvántartás[i] != „tulipán”) [
    i = i + 1
]
ha i < darab nyilvántartás [
    ki „Van tulipán. Sorszáma: ” + lánc(i + 1)
][
    ki „Nincs tulipán.”
]
```

Tehát feltételes ciklussal lépkedünk a lista elemein, amíg a lista végére nem érünk, vagy amíg nem találunk tulipánt. Amennyiben a lista vége előtt megálltunk, az azt jelenti, hogy volt tulipán. Ekkor pontosan az a virágágyás a tulipános, amelyiknél megálltunk. (Mivel a listák számozását 0-val kezdjük, ezért a kiíráskor hozzáadunk 1-et a sorszámhoz.)

A fenti ciklus helyett használható a lista adatszerkezet index() eljárása is, amely egyszerre visszaadja a keresett elem sorszámát (l. 87. oldal):

```
ha „tulipán” nyilvántartás-ban [
    ki „Van tulipán. Sorszáma: ” + lánc(nyilvántartás.index(„tulipán”) + 1)
][
    ki „Nincs tulipán.”
]
```

Feladat: Egy osztály orvosi vizsgálaton vett részt, amelyen a testmagasságukat is megmérték. Döntsd el a mérési eredményeket tartalmazó lista alapján, hogy van-e az osztályban 160 cm-nél magasabb diák, és ha igen, milyen magas? (A példában véletlenszerű értékeket fogok vizsgálni.)

```
n = egész véletlen 40  ; az osztály létszáma egy 40-nél kisebb egész szám
magasságok = [] ; létrehozom az üres magasságok listát
ismétlés n [
    ; hozzáadom a következő véletlenszerű magasságot:
    magasságok = magasságok + [100 + véletlen 70]
]
ki magasságok
 i = 0
amíg (i < darab magasságok) és (magasságok[i] <= 160) [
    i = i+1
```

```
]
ha i < darab magasságok [
    ki „Volt 160 cm-nél magasabb. Magassága: ” + lánc(magasságok[i])
][
    ki „Nem volt 160 cm-nél magasabb.”
]
```

A második példánkat már csak a kiírásnál kellett kiegészíteni. A feltételes ciklus most is vagy úgy állt meg, hogy a lista végére értünk, vagy pedig úgy, hogy talált egy 160 cm-nél magasabb diákot. Ha a második eset miatt állt meg, akkor az i éppen egy keresett magasságú ember sorszáma, aki tehát magasabb, mint 160 cm, így az ő magasságát írjuk ki.

Hasonló módosítást kell tennünk a harmadik feladatnál is.

Feladat: Egy rendezvény résztvevőinek neveit a nevek listában tároljuk. Készítsünk programot, amely eldönti, van-e a jelenlevők között „V” betűvel kezdődő nevű. Ha igen, hogy hívják?

A neveket egyesével olvassuk be a nevek listába:

```
nevek = []
név = be „Add meg a következő résztvevő nevét!”
amíg név != „” [
    nevek = nevek + [név]
    név = be „Add meg a következő résztvevő nevét!”
]
i = 0
amíg (i < darab nevek) és ( nevek[i][0] != „V” ) [
    i = i+1
]
ha i < darab nevek [
    ki „Volt V-vel kezdődő. Neve: ” + nevek[i]
][
    ki „Nem volt V-vel kezdődő.”
]
```

Egy érdekes keresési feladatot fogalmaz meg a most következő példa.

Feladat: Egy rendezvényen feljegyzik a résztvevők neveit a regisztrációjuk sorrendjében. Készítsünk programot, amely megadja, hogy van-e két ember, akiknek a neve ugyanazzal a betűvel kezdődik. Ha igen, akkor adjuk meg, kik ők?

Ennél a feladatnál a lista elemeit hasonlítjuk össze. Ehhez két indexre és két egymásba ágyazott feltételes ciklusra is szükségünk lesz. Az első index a regisztráció sorrendjében végighalad az összes személyen, kivéve a legkésőbb regisztrált

személyt. Az első indexszel jelölt személyt pedig a belső ciklus segítségével összehasonlítjuk minden később regisztrált személlyel, akiket a második index jelöl.

```
nevek = []
név = be „Add meg a következő résztvevő nevét!”
amíg név != „” [
    nevek = nevek + [név]
    név = be „Add meg a következő résztvevő nevét!”
]
i = 0
k = 1
amíg (i < darab(nevek) – 1 ) és (nevek[i][0] != nevek[k][0]) [
    amíg (k < darab nevek) és (nevek[i][0] != nevek[k][0])[
        k = k + 1
    ]
    ha nem (k < darab nevek)[
        i = i + 1
        k = i+1
    ]
]
ha i < darab(nevek) – 1 [
    ki „Volt 2 azonos kezdőbetűs név. Neveik: ” + nevek[i] + „ és ” + nevek[k]
][
    ki „Nem volt 2 azonos kezdőbetűs név.”
]
```

Itt valójában két kereséses feladatot kombináltunk egymással. A kisebb indexű névhez keresünk azonos kezdőbetűvel kezdődő nevet a nála nagyobb indexűek közül. Ha nem találtunk hozzá ilyet, akkor a következő indexű elemhez keresünk vele megegyező kezdőbetűset az őt követő nevek között.

Az amíg-os ciklus helyett használhatjuk a fut szerkezetet egy elágazással kombinálva. Ha megtaláltuk a feltételnek megfelelő elemet, akkor a kilép utasítással leállíthatjuk a ciklus futását. Ezt a megoldást mutatja be a következő példa, amelyet kiegészítünk a névszám változóval, amelyben tároljuk a nevek lista elemszámát.

```
névszám = darab nevek
fut i sor(névszám – 1)-ben [
    fut k sor(i + 1, névszám)-ban [
        ha nevek[i][0] == nevek[k][0] [ kilép ]
    ]
    ha nevek[i][0] == nevek[k][0] [ kilép ]
```

```
]
ha i < névszám – 1 [
    ki „Volt 2 azonos kezdőbetűs név. Neveik: ” + nevek[i] + „ és ” + nevek[k]
][
    ki „Nem volt 2 azonos kezdőbetűs név.”
]
```

Ugyanerre a feladatra mutatunk egy szótáras megoldást is a következő fejezetben a 127. oldalon.

Megszámolás

A *megszámolás* tételt azon feladatok megoldására használjuk, amikor valamilyen tulajdonságú elemek darabszámára vagyunk kíváncsiak.

- Hány tulipán van a virágoskertben?
- Hány piros virág van a kertben?
- Hány 1 szintes ház van a községben?
- Hány 10 betűsnél hosszabb nevű van a résztvevők között?
- Hány 160 cm-nél magasabb diák van az osztályban?
- Számoljuk meg, hány „C”-vel kezdődő nevű ember van a csoportban?
- Egy névsor alapján határozzuk meg, hogy hányféle kezdőbetűvel kezdődnek a névsor nevei.

Az ilyen feladatok megoldásának alapötlete, hogy a lista minden elemét végignézve, ha az elem megfelel a feltételnek, akkor növeljük a darabszámot tartalmazó változó értékét eggyel. Természetesen a darabszámot tartalmazó változónak a program elején 0 kezdőértéket kell adnunk. Lássunk néhány példát!

Feladat: Egy község házairól nyilvántartják, hogy hány emeletesek. Határozzuk meg, hogy a községben hány 1 szintes ház van!

Az értékeket most is véletlen számok segítségével adom meg. Arra figyeljünk, hogy a LibreLogóban a darab és a db változóneveket nem használhatjuk, mivel létező parancsok.

```
n = véletlen 100
házak = []
ismétlés n [
    házak = házak + [1 + egész(véletlen 4)]
]
ki házak
d = 0
```

```
fut ház házak-ban [
    ha ház == 1 [
        d = d + 1
    ]
]
ki lánc(d) + „ db 1 szintes ház van.”
```

Az utolsó példa szerintem különösen érdekes, ezért ennek megoldását is részletesen bemutatom.

Feladat: Egy névsor alapján határozzuk meg, hogy hányféle kezdőbetűvel kezdődnek a névsor nevei. (Feltételezhetjük, hogy legalább egy nevet tartalmaz a névsor.)

Ha a neveket névsorba rendezzük, akkor elegendő azt megszámolni, hogy hányszor változott a nevek első betűje az egymást követő neveknél. Azaz először rendezzük a neveket tartalmazó listát a rendez utasítás segítségével, majd a szomszédos nevek kezdőbetűit fogjuk vizsgálni. A szomszédos neveket elérhetjük ismétléses ciklussal és a hányadik változóval, de feltételes ciklussal is. Ekkor ügyeljünk rá, hogy ne felejtsük el az indexnek megfelelő kezdőértéket adni, és minden lépésben eggyel növelni, különben végtelen ciklust készítünk.

Mivel feltételeztük, hogy legalább egy név szerepel a névsorban, és így legalább egyféle kezdőbetűs név van, a darabszám kezdeti értéke 1 lesz. Ehhez adjuk hozzá, hogy hányszor változott az előzőhöz képest a kezdőbetű.

```
nevek = []
név = be „Add meg a következő résztvevő nevét!”
amíg név != „” [
    nevek = nevek + [név]
    név = be „Add meg a következő résztvevő nevét!”
]
nevek = rendez nevek
ki nevek
d = 1
ismétlés darab(nevek)–1 [
    ha nevek[hányadik–1][0] != nevek[hányadik][0] [
        d = d + 1
    ]
]
ki lánc(d) + „ különböző kezdőbetűs név van.”
```

Összegzés

Az ö*sszegzés* tételt alkalmazzuk azokban a feladatokban, amelyeknél az elemek összegére, szorzatára, átlagára, négyzetösszegére, harmonikus közepére, szövegek esetén kezdőbetűik összességére, átlagos hosszúságukra stb. vagyunk kíváncsiak. A feladatokban közös, hogy egy új elemre vonatkozó valamilyen értékét hozzávéve a korábbi elemekre már kiszámolt értékéhez, megkapjuk a kívánt eredményt. Ha például egy szavakat tartalmazó lista kezdőbetűit szeretnénk összegyűjteni, akkor ha 10 elemre már meghatároztuk a kezdőbetűket, akkor a 11. elem kezdőbetűjét hozzávéve a meglévőkhöz, a 11 elem kezdőbetűjét kapjuk eredményül.

- Számítsuk ki egy osztályban a gyerekek átlagos magasságát!
- Egy egércsaládban mindenki részt vett az elemózsiagyűjtésben. Egérpapa feljegyezte, hogy melyik egérke mennyi mogyorót gyűjtött. Ez alapján számítsuk ki, hogy mennyi elemózsiát gyűjtött összesen az egércsalád!
- Számítsuk ki az első n egész szám szorzatát (azaz $n!$ értékét)!
- Számítsuk ki egy szövegben a szavak átlagos hosszát!
- Adott n napra a napi legkisebb és legmagasabb hőmérséklet. Számítsuk ki az átlagos hőingadozás értékét!
- Pi-verseknek nevezzük azokat a verseket, amelyekben a szavak hossza rendre a π számjegyeivel egyezik meg. Készítsünk programot, amely egy pi-vers alapján megadja a π első valahány számjegyét!
- Egy programozási verseny több fordulóból áll. A diákok az első fordulóra kapott pontszámuk ¼-ét viszik tovább a következő fordulóba, amelyhez további pontokat szereznek, a harmadik fordulóba az addigi továbbvitellel kapott pontszámuk ¼-ét viszik tovább és így tovább. Pisti tudja, hogy mennyi pontot kapott minden fordulóban a feladataira. Számítsd ki, hogy mennyi pontot szerzett a versenyen összesítésben!

Lássuk az első példát.

Feladat: Számítsuk ki egy osztályban a gyerekek átlagos magasságát!

Ha az osztály gyerekeinek átlagos magasságát szeretnénk kiszámolni, elég összeadnunk minden gyerek magasságát, majd az eredményt elosztani a gyerekek számával. Az adatokat most is véletlenszerűen állítjuk elő. Az összeget kezdetben 0-ra állítjuk, majd ezt fogjuk növelni miden lépésben a következő gyerek magasságával, így megkapjuk a gyerekek magasságainak összegét. Végül ezt elosztjuk a gyerekek számával, amely a magasságokat tartalmazó lista elemszámával egyezik meg.

```
n = egész véletlen 40          ; az osztály létszáma egy 40-nél kisebb egész szám
magasságok = []                ; létrehozom az üres magasságok listát
ismétlés n [
    ; hozzáadom a következő véletlenszerű magasságot:
    magasságok = magasságok + [100 + véletlen 70]
]
ki magasságok
összeg = 0
fut magasság magasságok-ban [
    összeg = összeg + magasság
]
ki összeg/darab(magasságok)
```

Az egércsalád példáját meghagyom önálló feldolgozásra. Most nézzük inkább a harmadik példát.

Feladat: Számítsuk ki az első *n* egész szám szorzatát!

Az *n* értékét véletlenszerűen állítjuk elő. A megoldási ötlet nagyon hasonló az előző feladatéhoz. A szorzat korábbi értékét minden lépésben megszorozzuk a következő számmal. Azonban arra is kell figyelnünk, hogy a szorzat változónak az 1 kezdőértéket adjuk. Így ha egy számról van szó, akkor a szorzat valóban az az egy szám lesz. Ehhez szépen sorban hozzávéve a többi számot biztosan jó eredményt kapunk.

```
n = egész véletlen 10
számok = sor(1, n+1)
ki számok
szorzat = 1
fut szám számok-ban [
    szorzat = szorzat * szám [13]
]
ki szorzat
```

Most pedig a pi-vershez kapcsolódó feladatot beszéljük meg.

Feladat: Pi-verseknek nevezzük azokat a verseket, amelyekben a szavak hossza a π számjegyeivel egyeznek meg. Készítsünk programot, amely egy pi-vers alapján megadja a π első valahány számjegyét!

A példában Szász Pál matematikus 1952-ben készült pi-versét fogjuk feldolgozni. A verset karakterláncként adtuk meg. Ahhoz, hogy tovább dolgozhassunk vele, ki fogjuk nyerni a szavait egy listába a talál() függvény segítségével. A szám-

[13] Használható **szorzat *= szám** alakban is.

jegyeket karakterláncként fogjuk egymás után fűzni. Az első számjegy után egy tizedesvesszőt is be kell illesztenünk, hogy valóban a π-t kapjuk eredményül.

```
vers = „Nem a régi s durva közelítés, / Mi szótól szóig kijön / Betűiket számlálva. / Ludolph eredménye már, / Ha itt végezzük húsz jegyen. / De rendre kijő még tíz pontosan, / Azt is bízvást ígérhetem.”
szavak = talál(„\w+”, vers)
; ki szavak
szám = „”
fut szó szavak-ban [
    szám = szám + lánc(darab szó)
    ha hányadik == 1 [
        szám = szám + „,”
    ]
]
ki szám
```

Maximumkiválasztás

A m*aximumkiválasztás* tétel segítségével kiválaszthatjuk a lista valamilyen rendezés szerinti legnagyobb vagy legkisebb elemét.

- Határozzuk meg a legmagasabb diákot az osztályban!
- Hogy hívják a rendezvény leghosszabb nevű résztvevőjét?
- Adott *n* napra a napi legkisebb és legmagasabb hőmérséklet, melyik nap volt a legkisebb a hőingadozás?
- Egy kaszinóban mindenkinek van egy egyenlege, azaz nyilvántartják, hogy mennyi összeget nyert vagy vesztett az illető. Határozzuk meg, mennyi volt a legkisebb nyeremény, és kihez tartozik!

Az első feladatnál felmerül, hogy rendezzük a listát a magasságok értékei szerint, majd írjuk ki a lista utolsó elemét. Ekkor azonban csak a legmagasabb diák magasságát kapjuk meg, és arról nincs információnk, hogy ő melyik diák volt. Ezért használjuk a feladat megoldása során a *maximumkiválasztás* tételét. Az alapötlet, hogy kezdetben feltesszük, az első diák a legmagasabb, majd a második diáktól kezdve megvizsgáljuk, hogy az adott diák magasabb-e az eddig legmagasabbnak vélt diáknál. Ha igen, akkor ezentúl ő a legmagasabb, és hozzá viszonyítjuk a következő diákokat, amíg nem találunk még nála is magasabbat.

```
n = egész véletlen 40          ; az osztály létszáma egy 40-nél kisebb egész szám
magasságok = []                ; létrehozom az üres magasságok listát
ismétlés n [
```

```
    ; hozzáadom a következő véletlenszerű magasságot
    magasságok = magasságok + [100 + véletlen 70]
]
ki magasságok
legmagasabb = 0
fut magasság magasságok-ban [
    ha magasság > magasságok[legmagasabb] [
        legmagasabb = hányadik – 1
    ]
]
ki „A ” + lánc(legmagasabb+1) + „. diák a legmagasabb. ”
```

Az utolsó feladat megint különleges, ezért ezt külön megbeszéljük.

Feladat: Egy kaszinóban mindenkinek van egy egyenlege, azaz nyilvántartják, hogy mennyi összeget nyert vagy vesztett az illető. Határozzuk meg mennyi volt a legkisebb nyeremény, és kihez tartozik!

A feladat különlegessége, hogy a legkisebb nyeremény a nyilvántartás legkisebb pozitív értékeit jelenti. Előfordulhat, hogy nem is volt legkisebb nyeremény, mert minden résztvevő vesztett. Ezért meg kell keresünk az első személyt, aki nyert, majd elegendő az őt követő emberek nyereményei közül kiválasztani a legkisebb nyereményt.

```
n = véletlen 40
egyenleg = []
ismétlés n [
    egyenleg += [egész(véletlen 100)–50]
]
ki egyenleg
legkisebb = 0
amíg (legkisebb < darab egyenleg) és (egyenleg[legkisebb]<=0) [
    legkisebb = legkisebb + 1
]
 ha (legkisebb < darab egyenleg) [
    i = legkisebb+1
    amíg i < darab egyenleg [
        ha (egyenleg[i] > 0) és (egyenleg[legkisebb] > egyenleg[i])[
            legkisebb = i
        ]
         i = i + 1
    ]
     ki lánc(legkisebb + 1) + „., nyereménye: ” + lánc(egyenleg[legkisebb])
```

```
][
    ki „Nem volt legkisebb nyeremény, mert mindenki vesztett.”
]
```

Másolás

A m*ásolás* tételt olyan feladatokban használjuk, amikor egy lista minden eleméhez ki kell számítani egy új értéket, ezeket az új értékeket pedig egy listában szeretnénk tárolni.

- Gyakran találkozunk olyan versekkel, amelyekben a sorok első betűjét összeolvasva értelmes szöveget kapunk. Készíts programot, amely egy karakterláncokból lista első betűiből képez egy új listát!
- Adott *n* napra a napi legkisebb és legmagasabb hőmérséklet. Számítsuk ki minden napra az aznapi hőingadozás értékét!

Eddig még nem mutattam olyan feladatot, ahol két listából indultunk ki, ezért a második példát beszéljük meg részletesen.

Feladat: Adott *n* napra a napi legkisebb és legmagasabb hőmérséklet, számítsuk ki minden napra az aznapi hőingadozás értékét!

```
n = be „Hány napon át végezték a mérést?”
napi_minimumok = []
napi_maximumok = []
ismétlés n [
    napi_min = tört be „Napi minimum:”
    napi_max = tört be „Napi maximum:”
    napi_minimumok = napi_minimumok + [napi_min]
    napi_maximumok = napi_maximumok + [napi_max]
]
hőingadozás = []
ismétlés n [
    hőingadozás = hőingadozás + [napi_maximumok[hányadik–1] –
        napi_minimumok [hányadik – 1]]
]
ki hőingadozás
```

Az ilyen típusú feladatoknál létrehozunk egy üres eredménylistát, majd a kiindulási lista (listák) minden elemén végigmegyünk (ha csak egy lista van, akkor a fut szerkezettel), az elemből kiszámítjuk a szükséges értéket, majd hozzáfűzzük az eredménylistához.

Kiválogatás

A *kiválogatás* tétellel egy lista elemei közül egy másik listába kigyűjtjük a lista bizonyos tulajdonságú elemeit (a karakterláncoknál a talál() függvény hasonlót csinál).

- Válogassuk ki az osztályból a 160 cm-nél magasabbakat!
- Válogassuk ki a jelenlevők közül a „B”-vel kezdődő nevűeket!
- Adott *n* nap napi átlaghőmérséklete. Válogassuk ki a fagypont alatti hőmérsékletű napokat!
- Válogassuk ki egy szövegből az 5 betűs szavakat!

Az ilyen feladatok megoldása során végig kell mennünk az egész listán. Kezdetben egy üres eredménylistánk van, majd ehhez fűzzük hozzá a kiválasztandó elemeket, vagy azok sorszámát.

Feladat: Válogassuk ki az osztályból a 160 cm-nél magasabbakat!

```
n = egész véletlen 40
magasságok = []
ismétlés n [
    magasságok = magasságok + [100 + véletlen 70]
]
ki magasságok
magasak = []
fut magasság magasságok-ban [
    ha magasság > 160 [
        magasak = magasak + [hányadik – 1]
    ]
]
ki magasak
```

Ebben a példában a 160 cm-nél magasabb diákok indexét gyűjtöttük ki. Ha a magasságaikat is ki szeretnék írni, arra egy fut szerkezettel van lehetőségünk:

```
fut magas magasak-ban [
    ki lánc(magas+1) + „. diák, magassága: ” + lánc (magasságok[magas])
]
```

Mivel a többi példa megvalósításában nincs nagy különbség, ezért azt meghagyom az olvasónak.

Szétválogatás

A *szétválogatás* tétel hasonló a k*iválogatás* tételhez, csak itt a lista elemeit kettő vagy több listába válogatjuk szét valamilyen tulajdonságok szerint.

- Válogassuk szét az osztály tanulóit 160 cm-nél alacsonyabbakra és legalább 160 cm magasakra!
- Adott *n* nap napi átlaghőmérséklete. Válogassuk szét a napokat aszerint, hogy a hőmérséklet fagypont alatti, vagy annál nagyobb volt!
- Válogassuk szét a jelenlevőket a neveik kezdőbetűje alapján!
- Válogassuk szét egy szöveg szavait a szavak hossza alapján!
- Válogassuk szét a névnapokat tavaszi, nyári, őszi és téli névnapokra!

Az első két feladat megoldási módszere hasonló. Nem sokkal nehezebb a kiválogatás tételnél bemutatottnál, csak nem egy, hanem két listánk lesz. Vagy az egyikbe, vagy a másikba tesszük be a soron következő elemet. Az első feladat megoldása:

```
n = egész véletlen 40
magasságok = []
ismétlés n [
    magasságok += [100 + véletlen 70]
]
ki magasságok
magasak = []
alacsonyak = []
fut magasság magasságok-ban [
    ha magasság>160 [
        magasak += [hányadik – 1]
    ] [
        alacsonyak += [hányadik – 1]
    ]
]
ki magasak
ki alacsonyak
```

A harmadik és negyedik feladat jóval összetettebb, hiszen az elején még azt sem tudjuk, hogy hányfelé válogassuk szét az elemeket. Ezen feladatok megoldását a szótárakról szóló 18. fejezetben fogom bemutatni, mert szótárak használatával egyszerűbb a megvalósításuk.

Unió vagy egyesítés

Az u*nió* tétel a halmazoknál már megismert unió műveletet jelenti két listára, amely LibreLogóban a listák halmazzá konvertálásával általában egyszerűen megoldható. Azokban a feladatokban használjuk, amikor két listából akarjuk azokat az elemeket megkapni, amelyek legalább az egyik listában benne vannak.

Feladat: Egy programozási versenyen két programozási feladatra kellett megoldást küldeniük a diákoknak. Az egyik feladat megoldásait A tanár, a másikat B tanár pontozta. Mindegyikük készített egy listát azoknak a neveivel, akik a feladatot helyesen oldották meg. Akik legalább egy feladatra helyes megoldást adtak, oklevelet kapnak. Készítsd el az oklevelesek névsorát a két lista alapján!

```
névsor1 = [„Gipsz Jakab”, „Elektrom Ágnes”, „Ka Pál”, „Zsíros B. Ödön”]
névsor2 = [„Fa L. Ádám”, „Elektrom Ágnes”, „Akciós Áron”]
oklevelesek = lista(halmaz névsor1 | halmaz névsor2)
ki oklevelesek
```

Metszet

A m*etszet* tétel a halmazoknál már megismert metszet műveletet jelenti két listára, amely LibreLogóban a listák halmazzá konvertálásával általában egyszerűen megoldható. Azoknál a feladatoknál van szükségünk ennek alkalmazására, amikor két lista közös elemeit kell meghatároznunk.

Feladat: Egy programozási versenyen két programozási feladatra kellett megoldást küldeniük a diákoknak. Az egyik feladat megoldásait A tanár, a másikat B tanár pontozta. Mindegyikük készített egy listát azoknak a neveivel, akik a feladatot helyesen oldották meg. Akik mindkét feladatra helyes megoldást adtak, oklevelet kapnak. Készítsd el az oklevelesek névsorát a két lista alapján!

```
névsor1 = [„Gipsz Jakab”, „Elektrom Ágnes”, „Ka Pál”, „Zsíros B. Ödön”]
névsor2 = [„Fa L. Ádám”, „Elektrom Ágnes”, „Akciós Áron”]
oklevelesek = lista(halmaz névsor1 & halmaz névsor2)
ki oklevelesek
```

Feladatok

1) Olvass be egy listába egész számokat, majd készíts programot, amely megadja, van-e az elemek között négyzetszám!

2) Döntsük el egy egész számról, hogy prímszám-e!

3) Adjuk meg a legkisebb *n*-jegyű prímszámot!

4) Számoljuk meg hány „C”-vel kezdődő nevű ember van a csoportban!

5) Számítsuk ki egy osztályban a gyerekek átlagos magasságát!

6) Egy egércsaládban mindenki részt vett az elemózsiagyűjtésben. Egérpapa feljegyezte, hogy melyik egérke mennyi mogyorót gyűjtött. Ez alapján számítsuk ki, hogy mennyi elemózsiát gyűjtött az egércsalád összesen!

7) Számítsuk ki egy szövegben a szavak átlagos hosszát!

8) Adott *n* napra a napi legkisebb és legmagasabb hőmérséklet. Számítsuk ki az átlagos hőingadozás értékét!

9) Egy programozási verseny több fordulóból áll. A diákok az első fordulóra kapott pontszámuk ¼-ét viszik tovább a következő fordulóba, amelyhez további pontokat szereznek, a harmadik fordulóba az addigi továbbvitellel kapott pontszámuk ¼-ét viszik tovább és így tovább. Pisti tudja, hogy mennyi pontot kapott minden fordulóban a feladataira. Számítsd ki, hogy mennyi pontot szerzett a versenyen összesítésben.

10) Határozzuk meg a legmagasabb diákot vagy diákokat az osztályban!

11) Hogy hívják a rendezvény leghosszabb nevű résztvevőjét vagy résztvevőit?

12) Adott *n* napra a napi legkisebb és legmagasabb hőmérséklet. Mely nap vagy napokon volt a legkisebb a hőingadozás?

13) Válogassuk ki a jelenlevők közül a „B”-vel kezdődő nevűeket!

14) Adott *n* nap napi átlaghőmérséklete. Válogassuk ki a fagypont alatti hőmérsékletű napokat!

15) Válogassuk szét egy szöveg szavait a szavak hossza alapján!

Szótárak használata

A szótár egy különleges adatszerkezet, kulcs-érték párok halmaza. Például szótárban tárolhatjuk a szülő-gyerek kapcsolatot olyan formában, hogy a gyerek neve a kulcs, és az édesanyja neve az érték. A szótárban { } között vesszővel elválasztva soroljuk fel a kulcs-érték párokat, a párok közé kettőspontot teszünk.

```
szülő = {„Vince”: „Mónika”, „Fruzsi”: „Hajnalka”, „Panka”: „Mónika”, „Anna”: „Enikő”}
```

A szótár értékeit úgy kapjuk meg, ha a kulcsot használjuk „index”-ként:

```
ki szülő[„Vince”]
```

A szótár egy kulcsához tartozó értékét a listaelemekhez hasonlóan módosíthatjuk. A szótárakat is kiírathatjuk a ki utasítás segítségével.

```
szülő[„Anna”] = „Dóra”
ki szülő
```

Ha az értékadásnál olyan kulcsindexet adunk meg, amely még nem szerepel a szótárban, akkor új elemmel bővítjük a szótárunkat:

```
szülő[„Beni”] = „Mónika”
ki szülő
```

Így a szülő szótár tartalma a következő: {„Vince”: „Mónika”, „Beni”: „Mónika”, „Fruzsi”: „Hajnalka”, „Panka”: „Mónika”, „Anna”: „Dóra”}

A szótár elemein is végiglépkedhetünk a fut -ban/-ben szerkezet segítségével. Ilyenkor a megadott paraméter a kulcs értékeit fogja felvenni (véletlenszerű sorrendben):

```
fut kulcs szülő-ben [
    ki kulcs + „ szülője: ” + szülő[kulcs]
]
```

A fejezet hátralévő részében megmutatok két nehéz feladatot, amelyet szótárak segítségével egyszerűen meg lehet oldani.

Feladat: Készíts programot, amely a szöveggel megadott kétjegyű számot számjegyekkel írja ki!

Ha visszaemlékszünk, ennek a feladatnak a fordítottját már megvalósítottuk. Akkor egy listában tároltuk, hogy melyik számjegyet hogyan kell kiolvasni. Így a számjegy ismeretében csak az annyiadik indexű listaelemet kellett kiírnunk.

Most két szótárra lesz szükségünk, amelynek kulcsai az egyesek, illetve a tízesek szöveges változata lesz, míg az értékek a megfelelő számok lesznek. Először a megfelelő tízes helyi értékű szám nevének előfordulását nézzük meg a karakterláncokon is alkalmazható -ban/-ben segítségével. Miután eltároltuk a megfelelő számjegyet, töröljük a tízesek nevét a számból, így az egyeseket már egyszerűen kiolvashatjuk az egyesek szótárból. A számjegyeket karakterláncként fogjuk egymás után fűzni. Ha számként lesz rá szükségünk, akkor az egész() függvénnyel már könnyedén átalakíthatjuk.

```
egyesek = {„”:0, „egy”:1, „kettő”:2, „három”:3, „négy”:4, „öt”:5, „hat”:6, „hét”:7, „nyolc”:8, „kilenc”:9}
tízesek = {„tíz”:1, „húsz”:2, „tizen”:1, „huszon”:2, „harminc”:3, „negyven”:4, „ötven”:5, „hatvan”:6, „hetven”:7, „nyolcvan”:8, „kilencven”:9}
szám = be „Adj meg egy kétjegyű számot szöveggel!”
számjegyek = „”
fut tízes tízesek-ben [
    ha tízes szám-ban [
        számjegyek += lánc(tízesek[tízes])
        szám = cserél(tízes, „”, szám)
        számjegyek += lánc(egyesek[szám])
    ]
]
ki számjegyek
```

A másik példa egy már említett *szétválogatás* feladat lesz, amelynél előre nem tudjuk, hogy hányfelé kell a válogatást végeznünk.

Feladat: Válogassuk szét egy szöveg szavait a hosszuk alapján!

A példa megoldásban a már korábban ismertetett Szász Pál-féle pi-vers szavait válogatjuk szét. Először létrehozzuk a hossz nevű üres szótárat. Végiglépkedünk a szavakon, és ha az aktuális szó hossza még nem szerepel kulcsként a szótárban, akkor létrehozunk egy új kulcs-érték párt, amelynek kulcsa a szó hossza, az értéke a szót tartalmazó egyelemű lista. Ha az aktuális szó hosszának megfelelő kulcs már létezik a szótárban, akkor a hozzá tartozó listához hozzáfűzzük az új szót.

```
vers = „Nem a régi s durva közelítés, / Mi szótól szóig kijön / Betűiket számlálva. / Ludolph eredménye már, / Ha itt végezzük húsz jegyen. / De rendre kijő még tíz pontosan, / Azt is bízvást ígérhetem.”
szavak = talál(„\w+”, vers)
ki szavak
hossz = {}
fut szó szavak-ban [
```

```
        ha darab(szó) hossz-ban [
            hossz[darab(szó)] += [szó]
        ][
            hossz[darab(szó)] = [szó]
        ]
    ]
    ki hossz
```

Most pedig a keresésnél már tárgyalt feladatra mutatok be egy újabb megoldást szótár segítségével.

Feladat: Egy rendezvényen feljegyzik a résztvevők neveit a regisztrációjuk sorrendjében. Készítsünk programot, amely megadja, hogy van-e két ember, akiknek a neve ugyanazzal a betűvel kezdődik. Ha igen, akkor adjuk meg, kik ők!

A megoldás alapötlete, hogy egy szótárban gyűjtjük az érintett szavakat. A szótár kulcsai a szavak kezdőbetűi, az értékei pedig az adott betűvel kezdődő szavak listája. Ha már szerepel a soron következő kezdőbetű a szótár kulcsai között, akkor volt két azonos kezdőbetűs név. Ha így van, akkor az egyik szó az, amelyiknél megálltunk, a másik pedig ennek a szónak a kezdőbetűjével kezdődő szó a szótárban.

```
nevek = [„Anna”, „Bence”, „Cecília”, „Botond”]
nevek2 = {}
fut i nevek-ben [
    ha i[0] nevek2-ben [
        kilép
    ]
    nevek2[i[0]] = i
]
ha darab(nevek2) < darab(nevek) [
    ki „A két név: ” + i + „ és ” + nevek2[i[0]]
]
```

Feladatok

16) Adott két lista, az egyik a hónapok nevét, a másik a hónap napjainak számát tartalmazza. A két lista felhasználásával készíts szótárat, amelynek kulcsai a hónapok nevei, értékei a hónap napjainak a száma!

17) Az előző feladatban előállított szótár segítségével add meg, hogy az év *n*-edik napja milyen dátumnak felel meg!

18) Az előző feladatban létrehozott szótárt felhasználva készíts hónap naptárat rajzoló eljárást. Az adott hónapot, és hogy a hónap első napja a hét mely napjára esik, a billentyűzetről olvasd be! A naptár az alábbi minta szerint készüljön el.

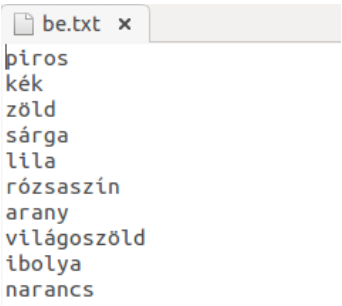
Január

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Adatok beolvasása fájlból

Az eddigi programjainknál az adatainkat beleírtuk a kódba, véletlenszerű értékeket állítottunk elő, vagy pedig beolvastuk őket a billentyűzetről. Ha nagyon sok adattal dolgozunk, célszerű azokat egy külön állományban elhelyezni. Ezt fogjuk tenni mi is, és megtanuljuk, hogyan tudjuk beolvasni egy szöveges állományból az adatainkat.

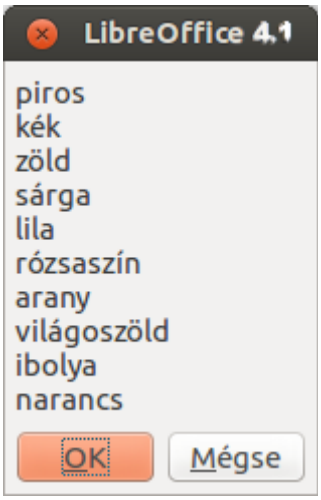
Természetesen most is valami konkrét feladattal kezdünk. Legyen a szöveges állományunk tartalma a következő:



Ezt a szöveges állományt bármilyen egyszerű szövegkezelővel elkészíthetjük, ilyen például *Ubuntun* és más GNU/Linux operációs rendszereken a *gedit*, vagy *Windows*-on a *Jegyzettömb*. Az állományt be.txt néven mentsük. Mentés közben állítsuk az állomány karakterkódolását UTF-8 (Unicode) kódolásúra, mert az ilyen állományokat kezeli ékezethelyesen a LibreLogo.

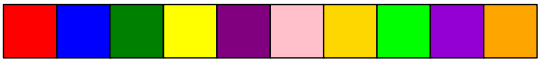
Miután a fájl tartalmát beolvastuk, el kell tárolnunk egy változóban. Ezt a változót most befájl-nak fogom nevezni, de természetesen más változónevet is adhatnánk neki. Ahhoz, hogy a fájlunkat elérjük, meg kell adnunk, hogy hova és milyen néven mentettük, azaz meg kell adnunk a fájl elérési útját és a fájl nevét. A példában Ubuntu operációs rendszeren a home könyvtár sajat_mappa nevű alkönyvtárában található a be.txt állomány. Az állomány tartalmának beolvasása és eltárolása a megfelelő változóba a következőképpen történik:

```
fájlhelye = „/home/sajat_mappa/be.txt”
befájl = open(fájlhelye). read()
ki befájl
```



Láthatjuk, hogy ilyenkor a befájl egy karakterlánc lesz, amelynek értéke a fájl tartalma. Ezt a szövegkezelő függvények segítségével bonthatjuk tovább feldolgozható formába, például a talál() függvény segítségével eltárolhatjuk egy listában a fájlban szereplő szavakat (színeket), ezután már használhatjuk a beolvasott értékeket további feladatok megoldására.

```
színek = talál(„\w+”, befájl)
fut szín színek-ben [
    töltőszín szín
    négyzet 20
    tollatfel jobbra 90 előre 20 balra 90 tollatle
]
```

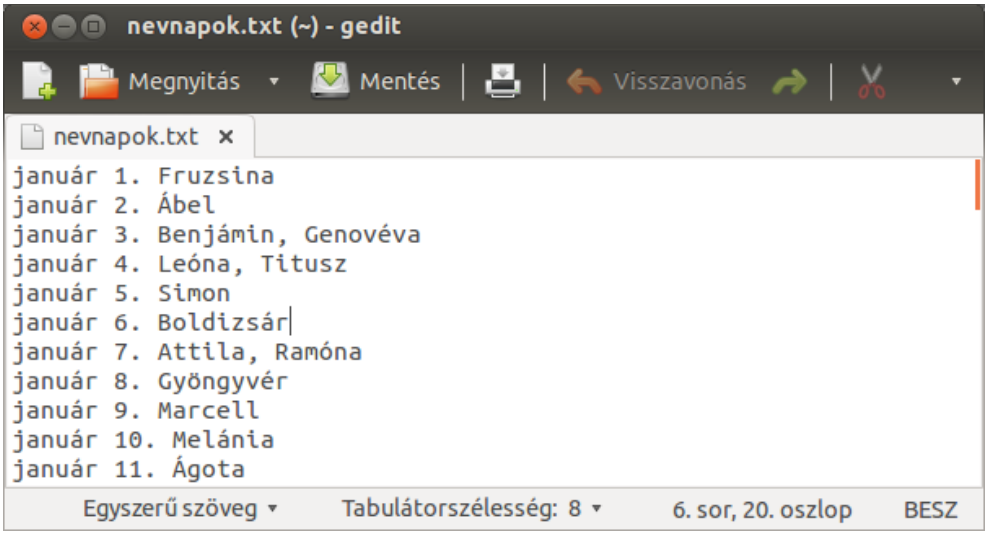


Az open() utasítással megnyitott állomány tartalma soronként is beolvasható a fut ciklussal. A beolvasott sorok végén található sortörés karaktert (\n) a strip() függvénnyel vágjuk le az előző példa módosított változatában:

```
fut szín open(„/home/sajat_mappa/be.txt”)-ben [
    töltőszín szín.strip()
    négyzet 20
    tollatfel jobbra 90 előre 20 balra 90 tollatle
]
```

Most pedig jöjjön a korábban említett névnapos feladat megoldása fájlból beolvasással.

Feladat: Adjuk meg az első olyan dátumot, amelyen „A”-val kezdődő név névnapja van. A névnapokat[14] a nevnapok.txt tartalmazza az alábbi formában:



Tehát minden sor tartalmaz egy dátumot és utána vesszővel elválasztva azokat a neveket, amelyeknek aznap van a névnapja. Az adatok dátum szerint rendezve vannak.

Az adatokat szótárba rendezzük oly módon, hogy a kulcsok a dátumok, az értékek a dátumhoz tartozó nevek karakterlánca lesz. Soronként beolvassuk a fájl tartalmát az akt_sor változóba. A sorokat a dátum végén lévő pontoknál vágjuk szét a dátumot tartalmazó kulcsra és a neveket tartalmazó karakterláncra a split() függvénnyel, amelynek paramétereként a pontot kell megadnunk. Mivel a szótárban a kulcs-érték párok rendezetlenül kerülnek be, ezért egy dátumok listát is létrehozunk, amelybe a beolvasások sorrendjében beletesszük a dátumokat.

```
dátumok = []
névnapok = {}
fut akt_sor open(„/home/sajat_mappa/nevnapok”)-ban [
    akt_sor = akt_sor.strip().split(„.”)
    dátumok += [akt_sor[0]]
    névnapok[akt_sor[0]] = akt_sor[1]
]
```

Most már rendelkezésünkre állnak a szükséges adatok. A feladat hátralévő részéhez a *kiválasztás* tétel megoldási elvét alkalmazzuk. A dátumokon lépkedünk

[14] A névnapok adatainak forrása *(utolsó elérés: 2013. október 29.)*: http://www.naptarak.com/naptarak_hu_nevnap_datum.html

végig, amíg egy olyan dátumhoz nem érünk, amelyen van „A”-val kezdődő névnap. Hogy az adott dátumon van-e „A”-val kezdődő névnap, a -ban/-ben művelettel ellenőrizzük.

```
i = 0
névnap = névnapok[dátumok[i]]
amíg nem „A” névnap-ban [
    i += 1
    névnap = névnapok[dátumok[i]]
]
ki dátumok[i]
```

Feladatok

19) A programozási tételeknél megbeszélt feladatokhoz készíts bemeneti fájlokat, és módosítsd a programokat úgy, hogy fájlból olvassák be a feldolgozandó adatokat!

Adatok fájlba írása

Ebben a fejezetben egyrészt megmutatom, hogyan tudjuk az elkészült képeinket valamilyen szabványos képformátumban elmenteni, másrészt pedig a szöveges állományba írással is megismerkedünk.

A vektorgrafikus képek leggyakoribb, a webböngészők által is ismert formátuma az SVG[15]. Az SVG állományba mentés a kép utasítás segítségével történik. Csak meg kell adnunk, hogy a képünket mely mappába, és milyen néven mentsük el.

```
eljárás kochpeano méret szint
    ha (szint==1)[
        előre méret
    ][
        kochpeano méret/3 szint–1
        ismétlés 3 [ kochpeano méret/3 szint–1 jobbra 120 ]
        balra 60
        ismétlés 2 [ kochpeano méret/3 szint–1 jobbra 120 ]
        balra 180
        kochpeano méret/3 szint–1
    ]
vége
tollszín „lila”
kép „kochpeano.svg” [
    kochpeano 200 4
]
```

Az utasítás hatására létrejött a kochpeano.svg állomány a megadott mappában.[16]

Animált SVG állományt is készíthetünk, ha várakozást építünk be a fraktálrajzoló eljárásunkba:

```
eljárás kochpeano méret szint
    ha (szint==1)[
        előre méret
        várj 5
```

[15] Az SVG a Scalable Vector Graphics rövidítése, a vektorgrafikus ábrák egy szabványos formátuma.

[16] Ha nem adunk meg elérési útvonalat, az állomány akkor is létrejön, és a dokumentummal megegyező mappába, vagy új, még nem mentett dokumentum esetén a felhasználó saját könyvtárába kerül.

```
    ][
        kochpeano méret/3 szint–1
        ismétlés 3 [ kochpeano méret/3 szint–1 jobbra 120 ]
        balra 60
        ismétlés 2 [ kochpeano méret/3 szint–1 jobbra 120 ]
        balra 180
        kochpeano méret/3 szint–1
    ]
vége
tollszín „lila”
kép „kochpeano_animalt.svg” [
    kochpeano 200 4
]
```

A LibreLogóval nem csak a képeinket, hanem a programjainkkal kiszámított értékeket is elmenthetjük egy szöveges állományban. Ehhez elegendő a megfelelő Python utasításokat használnunk. A fájlba íráshoz szintén szükség van egy változóra, ez az alábbi példában a kifájl nevű változó lesz. Az open() függvény segítségével létrehozzuk a megadott mappában a megadott nevű fájlt (ha már létezik az adott helyen ilyen néven fájl, akkor felülírja a tartalmát). A függvény második paramétere a „w” karakter lesz, amely arra utal, hogy a fájl kimeneti fájl lesz (a „w” az angol *write*, magyarul *írni* szó kezdőbetűje). Az adatokat a kifájl fájlváltozó saját write() függvényének meghívásával írjuk a kimeneti állományba. Az írás befejeztével be kell zárnunk az állományt a close() függvénnyel.

```
kifájl = open(„ki.txt”, „w”)
fut szín színek-ben [
    kifájl.write(szín+„\n”)
]
kifájl.close()
```

A write() függvény paramétere egyetlen karakterlánc lehet, ezért a számértékeket a kiírás előtt karakterlánccá kell alakítanunk. Ha több karakterláncot akarunk kiírni, akkor azt megtehetjük a write() ismételt meghívásával, ahogy ezt a fenti példában is tettük, de a + művelettel egy karakterlánccá is fűzhetjük kiírás előtt a karakterláncokat.

Feladatok

20) Mentsd el a korábban készült rajzaid közül a kedvenceidet SVG állományba is!

21) A programozási tételeknél megoldott feladatok eredményeit írd ki szöveges állományba!

22) Most már minden eszközt ismersz, hogy megoldd az érettségi programozási feladatait. Hajrá, láss neki a gyakorlásnak! A korábbi érettségi feladatok elérhetők a következő címen:
http://www.oktatas.hu/kozneveles/erettsegi/feladatsorok_vizsgatargyankent

Függelék – LibreLogo LibreOffice súgóoldal[17]

A LibreLogo egyszerű, honosított, Logo-szerű programozási környezet teknőc vektorgrafikával az informatika (programozás és szövegszerkesztés) oktatásához, kiadványszerkesztéshez és grafikai tervezéshez.

LibreLogo eszköztár

A LibreLogo eszköztár (Nézet→Eszköztárak→Logo) teknőcmozgató, programindítás és -leállítás, teknőc haza, képernyőtörlés, parancskiemelő/fordító ikonokat és egy beviteli mezőt (parancssor) tartalmaz.

Teknőcmozgató ikonok

Az „ELŐRE 10”, „HÁTRA 10”, „BALRA 15”, „JOBBRA 15” Logo parancsoknak felelnek meg. Az ikonokra kattintva a teknőc kijelölésre kerül, az oldalt a teknőc helyére gördítve.

Programindítás és -leállítás

Kattintson az „Indítás” ikonra a Writer dokumentum szövegtartalmának LibreLogo programként való indításához.

Kattintson a „Leállítás” ikonra a futó program leállításához.

Kiindulópont

Kattintson a teknőc kezdő pozíciójának és egyéb tulajdonságainak visszaállításához.

Képernyőtörlés

Kattintson a „Képernyőtörlés” ikonon a dokumentum rajzobjektumainak törléséhez.

[17] http://help.libreoffice.org/Writer/LibreLogo_Toolbar/hu

Programszerkesztő/parancskiemelés/fordítás

A „varázspálca” ikon kétoldalas nézetet állít be a programszerkesztéshez, kifejti és nagybetűsre alakítja a rövidített és kisbetűs Logo parancsokat a Writer dokumentumban. Módosítsa a dokumentum nyelvét (Eszközök → Beállítások → Nyelvi beállítások → Nyelvek → Nyugati), és kattintson az ikonon a Logo program kiválasztott nyelvre történő fordításához.

Parancssor

Nyomja le az Enter billentyűt a parancssorban a parancssor tartalmának végrehajtásához. A program leállításához használja a „Leállítás” ikont.

Tartsa nyomva az Entert a parancssor ismételt végrehajtásához, például a következő utasítássorozat megadása után:

ELŐRE 200 BALRA 89

A parancssor törléséhez kattintson rá háromszor, vagy nyomja meg a Ctrl-A kombinációt a korábbi parancsok kijelöléséhez és új parancsok beírásához.

Teknőc alapbeállítások grafikus felületen

A LibreLogo teknőce egy normál, rögzített méretű rajzobjektum. A szokásos módon áthelyezhető és forgatható az egér és a Rajzobjektum tulajdonságai eszköztár Forgatás ikonjának segítségével. A teknőc vonalvastagságának, vonal- és kitöltő színének módosítása a LibreLogo TOLLVASTAGSÁG, TOLLSZÍN és TÖLTŐSZÍN tulajdonságait állítja be.

Programszerkesztés

A LibreLogo rajzok és programok ugyanazt a Writer dokumentumot használják. A LibreLogo rajzterület a Writer dokumentum első oldala. Oldaltörés szúrható be a LibreLogo programok elé, és a megfelelő nagyítás állítható be a Logo eszköztár „varázspálca” ikonjára kattintva, további betűméret-beállítással kényelmes kétoldalas nézetet kapva a LibreLogo programozáshoz: bal (első) oldal a rajzterület, jobb (második) oldal a programszerkesztő.

LibreLogo programozási nyelv

A LibreLogo egy magyar nyelvű utasításokkal rendelkező, Logo-szerű programozási nyelv. Az oktatásban használt egyszerű Logo programok esetében visszafelé kompatibilis a régebbi Logo rendszerekkel, például

```
EZ háromszög:méret
ISMÉTLÉS 3 [
    ELŐRE:méret
    BALRA 120
]
VÉGE

háromszög 10 háromszög 100 háromszög 200
```

Eltérések a Logo programozási nyelvtől

- Vesszővel elválasztott listaelemek: HELY [0, 0]
- A programblokkok és a listák különböznek
- A programblokkok szóközzel vagy sortöréssel határoltak:
- ISMÉTLÉS 10 [ELŐRE 10 BALRA 36]
- Tapadó zárójel a listáknál: POZÍCIÓ [0, 0], nem pedig POZÍCIÓ [0, 0]
- Az egysoros függvénydeklarációk nem támogatottak (az EZ/TANULD/ELJÁRÁS és a VÉGE utasítás új sorban írandó).

Egyéb LibreLogo jellemzők

- A kettőspont nem kötelező a változónevek előtt:

```
EZ háromszög méret
ISMÉTLÉS 3 [ ELŐRE méret BALRA 120 ]
VÉGE
```

- A karakterláncok jelölése a helyesírást és a Python jelölést is támogatja.

```
KI "szó                    ; eredeti Logo jelölés
KI „Tetszőleges szöveg.”   ; helyesírás, Writer
KI 'Tetszőleges szöveg.'   ; Python jelölés
```

- Python lista- és karakterlánc-kezelés

```
KI „szöveg”[2]             ; „ö” kiírása
```

```
KI „szöveg”[1:3]                ; „zö” kiírása
```

- Python-szerű FUT ciklus
- Python-szerű változódeklaráció:

```
x = 15
KI x
```

- Nincsenek külön lekérdező függvények:

```
KI TÖLTŐSZÍN
p = HELY
KI p
ISMÉT 10 [ HELY TETSZŐLEGES HELY p ]
```

- Alternatív zárójelezés a függvényhívásokban:

```
EZ csillag méret szín
TÖLTŐSZÍN szín
ISMÉTLÉS 5 [
   BALRA 72 ELŐRE méret JOBBRA 144 ELŐRE méret
] TÖLT
VÉGE

csillag 100 „PIROS”
csillag (100, „ZÖLD”)
csillag(100, „KÉK”)
```

LibreLogo parancsok

Alapok

Kis- és nagybetűk megkülönböztetése

A parancs- és színnevek kis- és nagybetűvel is megadhatók:

```
KI „Szia világ!”
ki „Szia világ, újra!”
```

A változóneveknél viszont a kis- és nagybetű különbséget jelent:

```
a = 5
A = 7
KI a
KI A
```

Programsorok

A LibreLogo programsorok bekezdések a LibreOffice Writer dokumentumban. Egy programsor több parancsot is tartalmazhat:

```
KI „Szia, világ!” KI „LibreLogo”
```

Megjegyzések

A pontosvesszővel kezdett sorok vagy sorrészek megjegyzésnek számítanak a sor (bekezdés) végéig.

```
; megjegyzés
KI 5 * 5; megjegyzés
```

Programsorok több sorba tördelése

Egy programsor több sorba (bekezdésbe) törhető a hullámvonal karakter (~) segítségével.

```
KI „Ez egy nagyon hosszú” + ~
„figyelmeztetés.”
```

Teknőcmozgatás

ELŐRE (E, alternatív kompatibilitási név: CÍMKE2)

```
ELŐRE 10          ; előre 10 ponttal (1pt = 1/72 hüvelyk)
ELŐRE 10pt        ; mint előbb
ELŐRE 0,5in       ; előre 0,5 hüvelykkel (1 hüvelyk = 2,54 cm)
ELŐRE 1"          ; mint előbb
ELŐRE 1mm
ELŐRE 1cm
ELŐRE „felirat”   ; felírat kiírása a teknőc szöveg végére mozgatással (CÍMKE2)
ELŐRE [10, 200]   ; (x, y) pontba lép a teknőc helyének és irányának
                  ; megfelelő relatív koordináta-rendszerben
```

HÁTRA (H)

```
HÁTRA 10      ; hátra 10 ponttal
```

BALRA (B)

```
BALRA 90      ; fordulás 90°-ot az óramutató járásával ellentétesen
BALRA 90°     ; mint előbb
BALRA 3ó      ; mint előbb (órapozícióval)
BALRA 3h      ; mint előbb
BALRA TETSZŐLEGES ; fordulás véletlen irányba
```

JOBBRA (J)

```
JOBBRA 90     ; fordulás 90°-ot az óramutató járásának irányában
```

TOLLATFEL (TF)

```
TOLLATFEL     ; a teknőc ezután nem rajzol mozgás közben
```

TOLLATLE (TL)

```
TOLLATLE  ; a teknőc újra rajzol mozgás közben
```

HELY (POZÍCIÓ / XY)

```
HELY [0, 0]                ; fordulás és mozgás a bal felső sarokba
HELY OLDALMÉRET            ; fordulás és mozgás a jobb alsó sarokba
HELY [OLDALMÉRET[0], 0]    ; fordulás és mozgás a jobb felső sarokba
HELY TETSZŐLEGES           ; fordulás és mozgás az oldal véletlen pontjába
```

IRÁNY

```
IRÁNY 0                    ; fordulás északra
IRÁNY 12h                  ; mint előbb
IRÁNY TETSZŐLEGES          ; fordulás véletlen irányba
```

Egyéb teknőcparancsok

ELREJT (LÁTHATATLAN / ELREJTTEKNŐC / REJTTEK)

```
ELREJT                     ; a teknőc elrejtése (a LÁTHATÓ parancsig)
```

LÁTHATÓ

```
LÁTHATÓ                    ; teknőc megjelenítése
```

HAZA

```
HAZA                       ; visszatérés a kiinduló teknőcbeállításokra és -helyre
```

TÖRÖLKÉPERNYŐ (TR/ TÖRÖLKÉP/TÖRÖLRAJZLAP)

```
TÖRÖLKÉPERNYŐ              ; a rajzobjektumok eltávolítása a dokumentumból
```

TÖLT és ZÁR

```
TÖLT                       ; az aktuális vonalalakzat bezárása és kitöltése
ZÁR                        ; az aktuális vonalalakzat bezárása
```

Tollbeállítások

TOLLVASTAGSÁG (TV)

```
TOLLVASTAGSÁG 100                ; a vonalvastagság 100 pont
TOLLVASTAGSÁG TETSZŐLEGES        ; mint TOLLVASTAGSÁG VÉLETLEN 10
```

TOLLSZÍN (TSZ / VONALSZÍN)

```
; piros tollszín beállítása (színnév alapján, lásd a színkonstansokat)
TOLLSZÍN „PIROS”
TOLLSZÍN [255, 255, 0]           ; sárga szín beállítása (RGB lista)
TOLLSZÍN 0xffff00                ; sárga szín beállítása (hexadecimális kód)
TOLLSZÍN 0                       ; fekete szín beállítása (0x000000)
TOLLSZÍN TETSZŐLEGES             ; véletlen szín
; piros szín beállítása (színazonosító alapján, lásd a színkonstansokat)
TOLLSZÍN [5]
; láthatatlan tollszín a látható körvonal nélküli alakzatokhoz
TOLLSZÍN „LÁTHATATLAN”
TOLLSZÍN [255, 255, 0, 128]      ; átlátszó sárga szín beállítása (RGB lista)
TOLLSZÍN „~PIROS”                ; véletlen piros szín beállítása
```

TOLLÁTLÁTSZÓSÁG

```
TOLLÁTLÁTSZÓSÁG 80               ; az aktuális tollszín 80%-ban átlátszó
```

TOLLHEGY (VONALVÉG)

```
TOLLHEGY „NINCS”         ; extra vonalvég nélkül (alapértelmezés)
TOLLHEGY „KEREK”         ; lekerekített vonalvég
TOLLHEGY „NÉGYZET”       ; szögletes vonalvég
```

TOLLSAROK (VONALSAROK)

```
TOLLSAROK „KEREK”        ; lekerekített vonalcsatlakozás (alapértelmezés)
TOLLSAROK „HEGYES”       ; hegyes vonalcsatlakozás
TOLLSAROK „TOMPA”        ; tompa vonalcsatlakozás
TOLLSAROK „NINCS”        ; nincs vonalcsatlakozás
```

TOLLSTÍLUS (VONALSTÍLUS)

```
TOLLSTÍLUS „FOLYAMATOS”    ; folyamatos vonal (alapértelmezés)
TOLLSTÍLUS „PONTOZOTT”     ; pontozott vonal
TOLLSTÍLUS „SZAGGATOTT”    ; szaggatott vonal

; az egyéni pont-kötőjel mintázat a következő argumentumokat tartalmazó listával adható meg:
; – szomszédos pontok száma
; – pont hossza
; – szomszédos kötőjelek száma
; – kötőjel hossza
; – a pontok/kötőjelek távolsága
; – típus (elhagyható):
; 0 = a pontok téglalapok (alapértelmezés)
; 2 = a pontok négyzetek (a hosszak és távolságok a tollvastagsághoz képest értendők)

TOLLSTÍLUS [3, 1mm, 2, 4mm, 2mm, 2]          ; ...––...––...––
```

Kitöltési beállítások

TÖLTŐSZÍN (TLSZ)

```
TÖLTŐSZÍN „KÉK”                    ; kitöltés kék színnel, lásd még: TOLLSZÍN
TÖLTŐSZÍN „LÁTHATATLAN” KÖR 10     ; kitöltetlen kör
TÖLTŐSZÍN [„kék”, „piros”]         ; lineáris színátmenet a kék és piros között
TÖLTŐSZÍN [[255, 255, 255], [255, 128, 0]] ; fehér és narancssárga között
TÖLTŐSZÍN [„kék”, „piros”, 1, 0, 0]  ; visszavert (kék-piros-kék) színátmenet (a kötelezően megadandó forgatással és a szegéllyel), lehetséges értékek: 0-5 = lineáris, visszavert, sugaras, elliptikus, négyágú és négyoldalú színátmenet
TÖLTŐSZÍN [„kék”, „piros”, 0, 90, 20]  ; lineáris 20% szegéllyel, 90°-kal elforgatva a teknőc aktuális irányához képest
TÖLTŐSZÍN [„kék”, „piros”, 0, 90, 20, 0, 0, 200, 50] ; 200%-tól 50% fényességig
TÖLTŐSZÍN [TETSZŐLEGES, TETSZŐLEGES, 2, 0, 0, 50, 50] ; sugaras színátmenet véletlen színekkel, 50-50% vízszintes és függőleges pozícióban lévő középponttal
```

TÖLTŐÁTLÁTSZÓSÁG (ÁTLÁTSZÓSÁG)

TÖLTŐÁTLÁTSZÓSÁG 80 ; az aktuális töltőszín 80%-ban átlátszó
TÖLTŐÁTLÁTSZÓSÁG [80] ; lineáris átlátszósági gradiens 80%-tól 0%-ig
TÖLTŐÁTLÁTSZÓSÁG [80, 20] ; lineáris átlátszósági gradiens 80%-tól 20%-ig
TÖLTŐÁTLÁTSZÓSÁG [80, 20, 1, 90] ; visszavert átlátszósági gradiens 90°-kal elforgatva a teknőc aktuális irányához képest
TÖLTŐÁTLÁTSZÓSÁG [80, 20, 2, 0, 20, 50, 50] ; sugaras átlátszósági gradiens külső 80%-tól a belső 20%-ig, 20% szegéllyel és a középpont 50-50% vízszintes és függőleges pozíciójával.

TÖLTŐSTÍLUS

```
TÖLTŐSTÍLUS 0          ; kitöltés vonalkázás nélkül (alapértelmezés)
TÖLTŐSTÍLUS 1          ; fekete egyszeres vonalkázás (vízszintes)
TÖLTŐSTÍLUS 2          ; fekete egyszeres vonalkázás (45 fok)
TÖLTŐSTÍLUS 3          ; fekete egyszeres vonalkázás (–45 fok)
TÖLTŐSTÍLUS 4          ; fekete egyszeres vonalkázás (függőleges)
TÖLTŐSTÍLUS 5          ; piros metsző vonalkázás (45 fok)
TÖLTŐSTÍLUS 6          ; piros metsző vonalkázás (0 fok)
TÖLTŐSTÍLUS 7          ; kék metsző vonalkázás (45 fok)
TÖLTŐSTÍLUS 8          ; kék metsző vonalkázás (0 fok)
TÖLTŐSTÍLUS 9          ; kék háromszoros metsző vonalkázás
TÖLTŐSTÍLUS 10         ; fekete széles egyszeres vonalkázás (45 fok)

; egyéni vonalkázás a következő argumentumokat tartalmazó listával
; adható meg:
; – stílus (1 = egyszeres, 2 = dupla, 3 = tripla vonalkázás)
; – szín
; – távolság;
; – fok

TÖLTŐSTÍLUS [2, „ZÖLD”, 3pt, 15°]      ; zöld dupla vonalkázás (15 fok)
```

Rajzobjektumok

KÖR

```
KÖR 100                    ; kör alakzat rajzolása (átmérő = 100pt)
```

ELLIPSZIS

```
ELLIPSZIS [50, 100]              ; ellipszis rajzolása 50 és 100 átmérőkkel
ELLIPSZIS [50, 100, 2h, 12h]     ; ellipszisszektor rajzolása (a 2ó pozíciótól
                                 ; 12ó pozícióig)
ELLIPSZIS [50, 100, 2h, 12h, 2]  ; ellipszisszakasz rajzolása
ELLIPSZIS [50, 100, 2h, 12h, 3]  ; ellipszisív rajzolása
```

NÉGYZET

```
NÉGYZET 100                      ; négyzet alakzat rajzolása (méret = 100pt)
```

TÉGLALAP

```
TÉGLALAP [50, 100]               ; téglalap alakzat rajzolása (50×100pt)
TÉGLALAP [50, 100, 50]           ; téglalap rajzolása
```

PONT

```
PONT                 ; pont rajzolása a toll méretének és színének megfelelően
```

CÍMKE

```
CÍMKE „szöveg”               ; szöveg kiírása a teknőc pozíciójában
CÍMKE 'szöveg'               ; mint fent
CÍMKE "szöveg                ; mint fent (csak egyetlen szó)
CÍMKE [0, 0, „szöveg”]       ; mint fent, szöveg közepe a teknőc pozíciójában
CÍMKE [1, 1, „szöveg”]       ; szöveg bal alsó sarka a teknőc pozíciójában
CÍMKE [1, –1, „szöveg”]      ; szöveg bal felső sarka a teknőc pozíciójában
```

SZÖVEG

```
KÖR 10 SZÖVEG „szöveg”       ; az adott rajzobjektum szövegének beállítása
```

Betűbeállítások

BETŰSZÍN

```
BETŰSZÍN „ZÖLD”          ; betűszín beállítása
```

BETŰCSALÁD

```
BETŰCSALÁD „Linux Libertine G”                  ; betűcsalád beállítása
BETŰCSALÁD „Linux Libertine G:smcp=1”           ; betű tulajdonságainak (kis
                                                ; kapitális) beállítása
BETŰCSALÁD „Linux Libertine G:smcp=1&onum=1” ; kiskapitális és ugráló számok
```

BETŰMÉRET

```
BETŰMÉRET 12                                    ; 12 pontos méret beállítása
```

BETŰVASTAGSÁG

```
BETŰVASTAGSÁG „FÉLKÖVÉR”                        ; félkövér betűk beállítása
BETŰVASTAGSÁG „ÁLLÓ”                            ; normál vastagság beállítása
```

BETŰSTÍLUS

```
BETŰSTÍLUS „DŐLT”                               ; dőlt változat beállítása
BETŰSTÍLUS „ÁLLÓ”                               ; normál változat beállítása
```

KÉP

A KÉP a következőkre szolgál:

- alakzatok csoportosítása;
- új vonalalakzatok kezdése;
- SVG képek és SVG/SMIL animációk mentése;
- a pozíciók és vonalalakzatok konzisztenciájának megtartása a bal szélen.

Alakzatok csoportosítása

```
; KÉP [ LibreLogo_parancsok ]
KÉP [ ELŐRE 100 KÖR 100 ]     ; fa-szerű csoportosított alakzat
```

Lásd a „Csoport” fejezetet a LibreOffice Writer súgójában.

```
EZ fa hely
   TOLLATFEL HELY hely IRÁNY 0 TOLLATLE
   KÉP [ ELŐRE 100 KÖR 100 ]; fa-szerű csoportosított alakzat
VÉGE

; csoportosított alakzatok egy csoportosított alakzatban
KÉP [ fa [30, 50] fa [100, 50] ]
```

Új vonalalakzatok kezdése

```
KÉP                          ; új vonalalakzat kezdése
ELŐRE 10 KÉP ELŐRE 10        ; két vonalalakzat
```

SVG képek mentése

```
KÉP „példa.svg” [ KÖR 5 ]       ; kép mentése a dokumentum, vagy a
                                ; felhasználó könyvtárába
KÉP „Asztal/példa.svg” [ ELŐRE 100 KÖR 5 ] ; mint előbb, relatív útvonallal
KÉP „/home/felhasználó/példa.svg” [ KÖR 5 ]  ; kép mentése abszolút
                                                ; útvonallal (Unix/Linux)
KÉP „C:\példa.svg” [ KÖR 5 ] ; kép mentése abszolút útvonallal (Windows)
```

SVG/SMIL animációk mentése (rajzok VÁRJ utasítással)

```
KÉP „animáció.svg” [ KÖR 5 VÁRJ 1000 KÖR 99 ] ; animáció mentése (l. VÁRJ)
KÉP „animáció.svg” [ KÖR 5 VÁRJ 1000 KÖR 99 VÁRJ 2000 ]       ; mint előbb, de az
; utolsó objektum utáni VÁRJ az animáció ismétlését eredményezi 2 másodperc
; után a SMIL-t támogató böngészőkben
```

Konzisztencia a bal szélen

A kép segítségével megtarthatók a pozíciók és vonalalakzatok konzisztenciája a Writer bal szélén:

```
KÉP [ KÖR 20 HELY [–100, 100] KÖR 20 ]
```

Ciklusok

ISMÉTLÉS (ISM/ISMÉT, kompatibilitás: VÉGTELENSZER/VSZER)

```
; ISMÉTLÉS szám [ parancsok ]
ISMÉTLÉS 10 [ ELŐRE 10 BALRA 45 KÖR 10 ]     ; ismétlés tízszer

; a szám nem kötelező
ISMÉTLÉS [ HELY TETSZŐLEGES ]                ; végtelen ciklus
```

HÁNYADIK

Ciklusváltozó (a FUT és AMÍG ciklusokban).

```
ISMÉTLÉS 100 [ ELŐRE HÁNYADIK BALRA 90 ]
```

FUT -BAN/-BEN

Ciklus egy lista elemeire:

```
FUT i [1, 5, 7, 9, 11]-BAN [
    ELŐRE i
    BALRA 90
]
```

Ciklus karaktersorozat karaktereire:

```
FUT i „szöveg”-BEN [
    CÍMKE i
    ELŐRE 10
]
```

AMÍG

```
AMÍG IGAZ [ HELY TETSZŐLEGES ]               ; végtelen ciklus
AMÍG HÁNYADIK <= 10 [ ELŐRE 50 BALRA 36 ]    ; mint az ISMÉTLÉS 10 [ ... ]
```

KILÉP

A ciklus megállítása.

```
ISMÉTLÉS [                          ; végtelen ciklus
    HELY TETSZŐLEGES
    HA HÁNYADIK == 100 [ KILÉP ]    ; mint az ISMÉTLÉS 100 [ ... ]
]
```

ÚJRA

Ugrás a ciklus következő ismétlésére.

```
ISMÉTLÉS 100 [
    HELY TETSZŐLEGES
    HA HÁNYADIK% 2 != 0 [ ÚJRA ]
    KÖR 10                  ; körök rajzolása minden második pozícióba
]
```

Feltételek

HA

```
; HA feltétel [ IGAZ blokk ]
; HA feltétel [ IGAZ blokk ] [ HAMIS blokk ]

HA a < 10 [ KI „Kicsi” ]
HA a < 10 [ KI „Kicsi” ] [ KI „Nagy” ]
```

ÉS, VAGY, NEM

Logikai operátorok.

```
HA a < 10 ÉS NEM a < 5 [ KI „5, 6, 7, 8 vagy 9” ]
```

Eljárások

EZ (ELJÁRÁS / ELJ / TANULD), VÉGE

Új szó (vagy eljárás).

```
EZ háromszög
ISMÉTLÉS [ ELŐRE 100 JOBBRA 120 ] TÖLT
VÉGE

ISMÉTLÉS 10 [ háromszög TOLLATFEL HELY TETSZŐLEGES TOLLATLE ]
```

EREDMÉNY

A függvény visszatérési értéke.

```
EZ véletlenbetű
EREDMÉNY VÉLETLEN „aábcdeéfghiíjklmnoóöőpqrstuúüűvwyz”
VÉGE

; 3 véletlen betűs karakterlánc kiírása
KI véletlenbetű + véletlenbetű + véletlenbetű
```

STOP (VISSZATÉR)

Visszatérés az eljárásból.

```
EZ példa szám
   HA szám < 0 [ STOP ]
   KI GYÖK szám                    ; négyzetgyök kiírása
]
példa 100
példa –1                                ; kimenet és hiba nincs
példa 25
```

Alapértelmezett változók

TETSZŐLEGES (TETSZ)

A színek stb. alapértelmezett véletlen értéke.

```
TOLLSZÍN TETSZŐLEGES                ; véletlen tollszín
```

IGAZ

Logikai érték.

```
AMÍG IGAZ [ HELY TETSZŐLEGES ]      ; végtelen ciklus
```

KI IGAZ; igaz kiírása

HAMIS

Logikai érték.

```
AMÍG NEM HAMIS [ HELY TETSZŐLEGES ]     ; végtelen ciklus
KI HAMIS                                ; hamis érték kiírása
```

OLDALMÉRET

```
; az oldalméretek pontban megadott listájának, például: [595,30, 841,89]
KI OLDALMÉRET
```

PI/π

```
KI PI                                   ; 3,14159265359 kiírása
```

Adatbevitel/kiíratás

KI (KIÍR)

```
KI „szöveg                              ; a „szöveg” kiírása párbeszédablakba
KI 5 + 10                               ; 15 kiírása
```

BE

```
; karakterlánc bekérése és kiírása párbeszédablak használatával
KI BE „Bemeneti érték?”
; egyszerű számológép
KI TÖRTSZÁM (BE „Első szám?”) + TÖRTSZÁM (BE „Második szám?”)
```

VÁR (VÁRJ)

```
VÁR 1000            ; várakozás 1000 ezredmásodpercig (1 mp)
```

GLOBÁLIS (GLOBÁLISVÁLTOZÓ/GLOBVÁL)

Eljárásokban használt globális változók megadása.

```
GLOBÁLIS névjegy
névjegy = „LibreLogo”

EZ példa
    KI névjegy
    GLOBÁLIS névjegy                ; új érték hozzáadása
    névjegy = „a globális változó új értéke”
VÉGE

példa
KI névjegy
```

Függvények

VÉLETLEN (VÉLETLENSZÁM/VSZÁM/KIVÁLASZT)

```
KI VÉLETLEN 100             ; véletlen lebegőpontos szám kiírása (0 <= x < 100)
KI VÉLETLEN „szöveg”        ; a „szöveg” véletlen betűjének kiírása
KI VÉLETLEN [1, 2]          ; a lista véletlen elemének kiírása (1 vagy 2)
```

EGÉSZSZÁM (EGÉSZ)

```
KI EGÉSZSZÁM 3,8            ; 3 kiírása (a 3,8 egész része)
KI EGÉSZSZÁM VÉLETLEN 100 ; véletlen egész szám kiírása (0 <= x < 100)
KI EGÉSZSZÁM „7”            ; a karakterlánc paraméter egésszé konvertálása
```

TÖRTSZÁM (TÖRT)

```
; a karakterlánc paraméter törtszámmá konvertálása
KI 2 * TÖRTSZÁM „5.5”       ; 11,0 kiírása
```

KARAKTERLÁNC (LÁNC)

```
; a szám paraméter karakterlánccá alakítása
KI „Eredmény: ” + LÁNC 5    ; „Eredmény: 5” kiírása
```

```
KI 10 * LÁNC 5                  ; 5555555555 kiírása
```

GYÖK

```
KI GYÖK 100                     ; 10 kiírása, amely a 100 négyzetgyöke
```

SIN

```
KI SIN 90 * PI/180              ; 1,0 kiírása (90° szinusza radiánban)
```

COS

```
KI COS 0 * PI/180               ; 1,0 kiírása (0° koszinusza radiánban)
```

LOG10

```
KI LOG10 100                    ; 2,0 kiírása (100 tízes alapú logaritmusa)
```

KEREKÍTÉS (KEREK)

```
KI KEREK 3,8                    ; 4 kiírása (3,8 kerekítve)
KI KEREK VÉLETLEN 100           ; véletlen egész szám (0 <= x <= 100) kiírása
```

ABSZOLÚTÉRTÉK (ABSZ)

```
KI ABSZOLÚTÉRTÉK –10            ; 10 kiírása, amely a –10 abszolút értéke
```

DARAB (DB / ELEMSZÁM)

```
KI DARAB „szöveg”               ; 6 kiírása, amely a „szöveg” karaktereinek száma
KI DARAB [1, 2, 3]              ; 3 kiírása, amely a lista mérete
```

HALMAZ

```
; lista Python halmazzá alakítása
KI HALMAZ [4, 5, 6, 6]          ; a {4, 5, 6} kiírása
KI HALMAZ [4, 5, 6, 6] | HALMAZ [4, 1, 9]          ; a {1, 4, 5, 6, 9} kiírása, unió
```

```
KI HALMAZ [4, 5, 6, 6] & HALMAZ [4, 1, 9]      ; a {4} kiírása, metszet
KI HALMAZ ([4, 5, 6, 6]) - HALMAZ [4, 1, 9]    ; a {5, 6} kiírása, különbség
KI HALMAZ [4, 5, 6, 6] ^ HALMAZ [4, 1, 9]      ; a {1, 5, 6, 9} kiírása,
                                               ; szimmetrikus különbség
```

SOR

```
; Python-szerű listagenerálás
KI SOR 10                          ; a [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] kiírása
KI SOR 3 10                        ; a [3, 4, 5, 6, 7, 8, 9] kiírása
KI SOR 3 10 3                      ; a [3, 6, 9] kiírása
FUT i SOR 10 50 10-BEN [           ; ciklus [10, 20, 30, 40] elemekre
    ELŐRE i
    BALRA 90
]
```

LISTA

```
; egy lista ismétlődő elemeinek eltávolítása halmaz és lista átalakításokkal
KI LISTA (HALMAZ [1, 3, 5, 5, 2, 1])      ; [1, 3, 5, 2] kiírása
```

FIX

Átalakítás Python tuple (nem módosítható lista) adattípussá

```
KI FIX [4, 5]
```

RENDEZ

Rendezett lista visszaadása.

```
KI RENDEZ [5, 1, 3, 4]             ; [1, 3, 4, 5] kiírása
```

CSERÉL

Karaktersorozatok cseréje reguláris kifejezés minták használatával.

```
KI CSERÉL („s”, „S”, „szöveg”)  ; a „Szöveg” kiírása, az „s”-t lecserélve „S”-re
KI CSERÉL („(.)”, „\\1\\1”, „szöveg”)     ; „sszzöövveegg” kiírása, minden karaktert
                                          ; megkettőzve
```

KERES

Karaktersorozatok keresése reguláris kifejezés minták használatával.

HA KERES („\w”, szó) [KI „Van betű a szóban.”]

TALÁL

A bemeneti karakterláncban a megadott reguláris kifejezésre illeszkedő összes karaktersorozat megkeresése.

```
; a [„Kutyák”, „macskák”] kiírása, a szavak listája.
KI TALÁL(„\w+”, „Kutyák, macskák.”)
```

MIN

```
KI MIN [1, 2, 3]    ; 1 kiírása, amely a lista legkisebb eleme
```

MAX

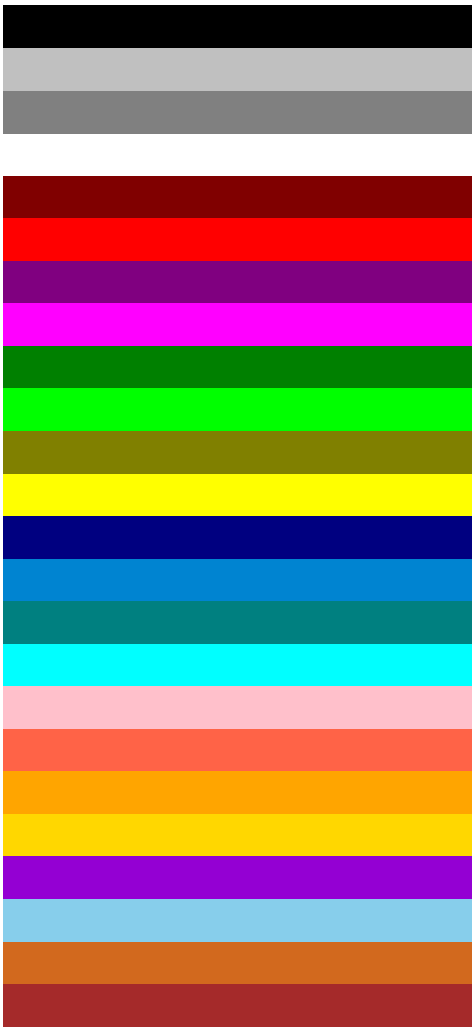
```
KI MAX [1, 2, 3]    ; 3 kiírása, amely a lista legnagyobb eleme
```

Színkonstansok

```
TOLLSZÍN „EZÜST”          ; színmegadás névvel
TOLLSZÍN [1]              ; színmegadás azonosítóval
TOLLSZÍN „~EZÜST”         ; véletlen ezüst szín
TOLLSZÍN 0xFF0000         ; piros
TOLLSZÍN 0x80FF0000       ; félig áttetsző piros
TOLLSZÍN [255, 0, 0]      ; piros
TOLLSZÍN [255, 0, 0, 128] ; félig áttetsző piros
```

Színkonstansok táblázata

Azonosító	Név
0	FEKETE
1	VILÁGOSSZÜRKE/EZÜST
2	SZÜRKE
3	FEHÉR
4	SÖTÉTBARNA
5	PIROS/VÖRÖS
6	LILA
7	BÍBOR/CIKLÁMEN
8	ZÖLD
9	VILÁGOSZÖLD
10	OLAJZÖLD
11	SÁRGA
12	SÖTÉTKÉK
13	KÉK
14	KÉKESZÖLD
15	CIÁNKÉK/CIÁN
16	RÓZSASZÍN
17	VILÁGOSPIROS
18	NARANCSSÁRGA/NARANCS
19	ARANYSÁRGA/ARANY
20	IBOLYAKÉK/IBOLYA/VIOLA
21	ÉGSZÍNKÉK/VILÁGOSKÉK
22	VILÁGOSBARNA
23	BARNA
24	LÁTHATATLAN



Ajánlott hivatkozások

1. LibreLogo honlap és hírportál: http://www.librelogo.org.

2. Németh László: *LibreLogo*, FSF.hu Alapítvány, 2013: http://www.numbertext.org/logo/logofuzet.pdf.

3. Wikipédia képek LibreLogo forráskóddal:http://commons.wikimedia.org/wiki/Category:Images_with_LibreLogo source_code.

MAGYARY PROGRAM

Nemzeti Fejlesztési Ügynökség
www.ujszechenyiterv.gov.hu
06 40 638 638

MAGYARORSZÁG MEGÚJUL

A projektek az Európai Unió
támogatásával valósulnak meg.