Mi az a programozás?

Mi a program?

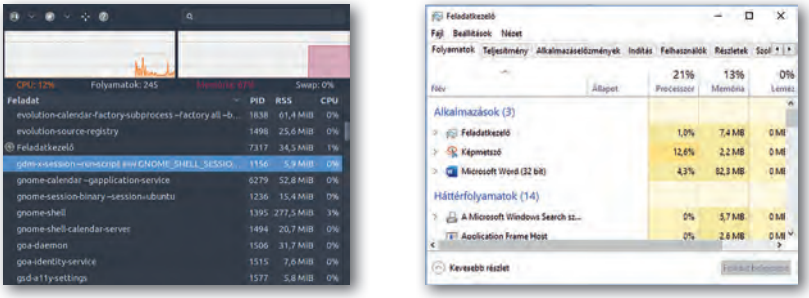
A számítógép, a telefon, az összes olyan eszközünk, amiben valamilyen számítógép van, önmagában képtelen ellátni azt a feladatot, amire készült. Csak egy darab „vas” – azaz hardver. Csak akkor képes igazán működni, ha fut rajta egy (vagy sok) program, alkalmazás – azaz szoftver. Szoftver, program, alkalmazás – nagyjából ugyanazt jelenti: azt a programozó, szoftverfejlesztő által megírt valamit, ami elmondja a hardvernek, hogy mikor mit csináljon.

Program mondja el a kenyérsütőnek, hogy meddig gyúrja a tésztát, meddig hagyja dagadni, és mikor kezdje sütni, mennyire legyen meleg a fűtőszál, és hányat sípoljon a sütő, amikor kész a kenyér. Program mondja el a mosógépnek, hogy mikor és mennyi vizet szívjon be, mennyire melegítse meg, meddig forogjon benne a dob, és meddig kell centrifugáznia. Ezek a számítógépek egyetlen programot futtatnak.

Az informatikaórán bennünket jobban érdekelnek a hagyományos értelemben vett számítógépek (laptopok, asztali gépek, szerverek) és a mobileszközök. Ezek csak bekapcsoláskor futtatnak egyetlen programot, ami azt mondja el nekik, hogy honnan és hogyan kell betölteniük a „fő” programjukat: az operációs rendszert. A többi program (a böngésző, az üzenetküldő, a játék, a képszerkesztő, a szövegszerkesztő, a filmvágó stb.) pedig az operációs rendszerből, annak felügyelete alatt indul el, akár úgy, hogy rákattintunk az egérrel vagy rábökünk az ujjunkkal az indítóikonjára, akár automatikusan.

Hol vannak a programok?

Az elindított, azaz futó programok a számítógép memóriájában vannak. Nemcsak a program van itt, hanem az általa éppen használt adatok is: a szövegszerkesztő által szerkesztett szöveg, a képszerkesztőbe betöltött kép. Az operációs rendszerünk feladatkezelőjében megnézhetjük az épp futó programokat. Látjuk, hogy a legtöbbet nem mi indítottuk el, sőt nem is látjuk őket – a háttérben futnak.

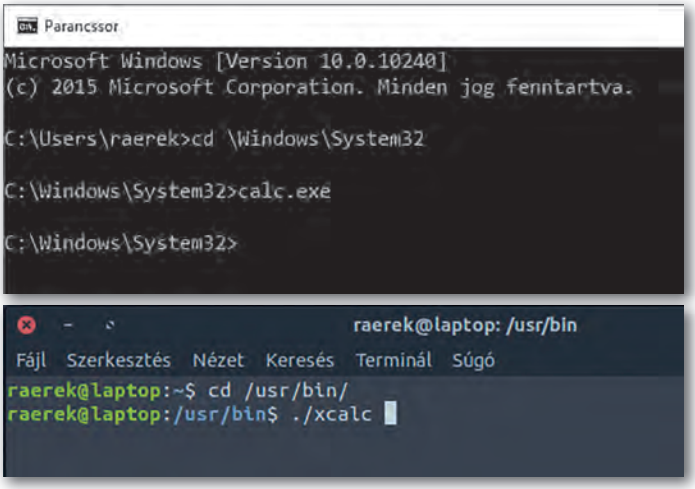


▶ Egy Linuxon futó feladatkezelő és a Windows 10 feladatkezelője – Indítsunk el egy programot, keressük meg a feladatkezelőben, és állítsuk meg innen!

Amíg a programot nem indítjuk el, a számítógép háttértárán, azaz a laptop SSD-jén, az asztali gép vagy szerver winchesterén, vagy a telefon memóriakártyáján van, ugyanolyan fájlként, mint a képek, zenék vagy szövegek. Az egyszerű programok egyetlen fájlból állnak, az összetettek sokszor nagyon sokból.

A grafikus felületű operációs rendszerek elterjedése előtt (az 1990-es évekig) a programokat úgy indítottuk el, hogy a parancssoros felületben beléptünk abba a mappába (könyvtárba), amelyikben a programunk volt, és beírtuk a program nevét. A módszer ma is működik, bár többnyire csak a számítógépekhez jobban értő emberek, rendszergazdák, rendszermérnökök és szoftverfejlesztők használják. Lévén e fejezet célja épp az, hogy kicsit mi is szoftverfejlesztők legyünk, ismerkedjünk meg ezzel a módszerrel!

1. Nyissunk a gépünkön parancssoros felületet: Windowson indítsuk el a Parancssor nevű alkalmazást, macOS-en és Linuxon pedig valamelyik terminált!
2. „`cd`” parancsokkal lépkedjünk abba a mappába, ahol a programfájl van (minden sor begépelése után Entert nyomunk)!
3. Írjuk be a program nevét, és nyomjuk meg az Entert!



► Program indítása parancssorból Windows 10-en és Ubuntu Linuxon

A program ilyenkor betöltődik a számítógép memóriájába, és a benne lévő utasítások végrehajtódnak, azaz a program futni kezd.

Mi van egy programfájlban?

Ha már úgyis a parancssorban, az imént elindított programunk mappájában vagyunk, adjuk ki

- Windowson a `type`
- macOS-en és Linuxon a `cat`

parancsot, és írjuk utána a programfájl nevét (például `type calc.exe`, `cat xcalc`)! Rengeteg krikszkrakszot ír a parancssori ablakba a gép. Ha némileg hihetetlen is, a számítógép ezt érti, ebből tudja, hogy mit kell csinálnia. Ez a program egyik alakja, az úgynevezett *gépi kódú* program, amely most a képernyőn karakterek formájában jelenik meg.

Szerencsére a legtöbb szoftverfejlesztőnek nem így kell megfogalmaznia a gép teendőit. Rendelkezésünkre állnak programozási nyelvek, azaz az angol nyelv szavait használó magasabb szintű nyelvek. Ilyen például a C, a C++, a C#, a Pascal, a Ruby, a Go, a Perl, a JavaScript, a Java, és még sorolhatnánk. Ilyen a mi könyvünkben használt **Python** is. A szoftverfejlesztő többnyire valamelyik ilyen programozási nyelvben készíti el a program **forráskódját**.

Egy egyszerű forráskódot többé-kevésbé már most is értelmezni tudsz. Mit csinál az alábbi, Python nyelvű program?

```
print('Üdv néked!')
évek_száma = input('Hány éves vagy?')
évek_száma = int(évek_száma)
if évek_száma < 14:
    print('Jé, hogyhogy már középiskolás vagy?')
else:
    print('Egy év múlva', évek_száma+1, 'éves leszel.')
```

Nos, ilyen és ehhez hasonló programokat fogunk mi is írni az elkövetkezendő órákon. Látjuk, hogy az angol szavak mellett van még a programban írásjel, műveleti jel, zárójel – ezek mind a program részei, nem hagyhatók el. A Pythonban szerepe van annak is, hogy a sor elején kezdődik-e egy programsor, vagy bentebb.

Természetesen ezt a programkódot a számítógép ebben a formában nem érti, és nem tudja futtatni. A fenti forráskódot egy másik program előbb gépi kóddá alakítja, és a gép processzora a gépi kódot értelmezve futtatja a programunkat.

Feladatok

1. Az alábbi, C++-nyelvű kód pontosan ugyanazt csinálja, mint a fenti Python nyelvű. Keressük meg a hasonlóságokat, mutassunk rá a különbségekre!

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Üdv néked!" << endl;
    int evek_szama;
    cout << "Hány éves vagy?";
    cin >> evek_szama;
    if (evek_szama < 14 ) {
        cout << "Jé, hogyhogy már középiskolás vagy?";
    } else {
        cout << "Egy év múlva " << evek_szama+1 << "éves leszel." << endl;
    }
    return 0;
}
```

2. Rakjunk össze egy másik programot az alábbi részletekből!

Csak ennyiből áll egy szoftverfejlesztő munkája?

Nos, igen, a fejlesztői munka legismertebb része az új programok írása.

Sokkal többen vannak azok, akik meglévő programokat alakítgatnak át az új követelményeknek megfelelően (nekik köszönhetők például a telefonjainkra letöltendő frissítések).

Van, aki azzal foglalkozik, hogy egy már meglévő program fusson másféle gépen is – ez néhány esetben gyorsan megoldható, más programoknál nehéz és kimerítő feladat.

Vannak, akik azért dolgoznak, hogy egy meglévő program legyen gyorsabb.

Van, aki programokat tesztel: megnézi, hogy biztosan jól működnek-e mindenféle helyzetben.

Van, aki azzal foglalkozik, hogy programot, szoftverrendszert tervez. Ő már nem ír kódot, hanem azért felel, hogy a szoftver különböző részei minél jobban tudjanak együtt dolgozni.

Van, aki biztonsági ellenőrzést végez programokon, például azért, hogy számítógépes bűnözők ne tudják a banki szoftverekkel átutaltatni a pénzünket másik számlára.

A következő leckében mi is megírjuk első programunkat.

Kérdések

1. Mik azok a számítógépes vírusok?
2. Milyen egyéb feladatok merülhetnek fel egy szoftver elkészítésekor? Milyen képzettségű munkatársai vannak a szoftver fejlesztőjének?
3. Milyen kép él benned a programozókról? Milyen előítéletek kapcsolódnak hozzájuk?
4. Milyen világszerte ismert oldalakon foglalkoznak programozási kérdések megválaszolásával? Hány programozással kapcsolatos videó készül naponta?

Első programjaink

A programozási környezet

A programozási környezet arra való, hogy benne írjuk meg programjainkat, használatával a programot gépi kódúvá alakítsuk, és tesztelhessük, dokumentálhassuk a kész programot. Ezek közül a legfontosabb a gépi kódúvá alakítás, a többit vagy meg tudjuk oldani egy adott környezetben, vagy nem.

A programozási környezetet telepítenünk kell a gépünkre. A Python nyelv környezete a `python.org` webhelyről tölthető le, méghozzá – lévén a Python szabad szoftver – ingyenesen, bárki számára. Egészen biztosan találunk gépünknek és operációs rendszerünknek megfelelő változatot. Gépigénye nagyon kicsi, azaz bátran telepítsük öregebb, kisebb teljesítményű gépekre is.

Legelső programunk

Az első programunkat egy egyszerű szerkesztőben írjuk meg, ilyen például a Windows Jegyzettömbje. Indítsuk el, és gépeljük bele ezt az egyetlen sort:

```
print('Szia.')
```

Ha Linuxon vagy macOS alatt dolgozunk, akkor valamennyi Python programunk legeslegelső sora kötelezően

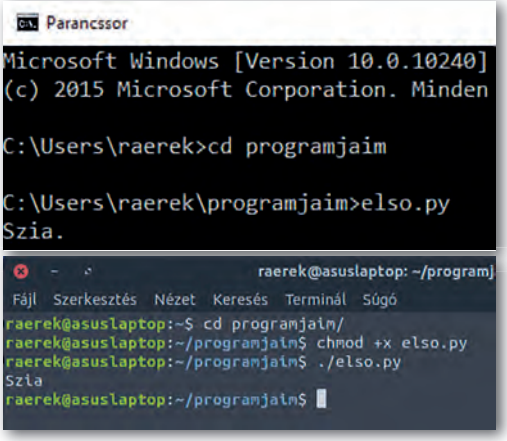
```
#!/usr/bin/env python3
```

legyen. Erre Windows alatt nincs szükség.

Mentsük el a fájlunkat – érdemes a programjainknak létrehozni egy mappát –, és figyeljünk rá, hogy a fájl kiterjesztése `.py` legyen, azaz a fájl teljes neve legyen például `elso.py`. Általában egy szóból álló és ékezetmentes neveket szoktunk programnévként használni. Egyes rendszerek a szóköz és az ékezetes betűk használatára érzékenyek. A `.py` kiterjesztés az állomány Python-forrásállomány jellegére utal.

A programokat mindig egyszerű szerkesztőkben írjuk, sohasem „igazi”, sok formázási és más funkciót tartalmazó szövegszerkesztőben. Ennek az a magyarázata, hogy a Word vagy a LibreOffice Writer a fájlba nemcsak azt menti, amit beleírtunk, hanem sok egyebet is, például a formázásokat.

Nyissunk parancssort, és a múlt alkalommal használt `cd` paranccsal lépjünk abba a mappába, ahova a fájlt mentettük. Linuxon és macOS-en még futtathatóvá kell tennünk a fájlt a `chmod +x elso.py` parancs kiadásával. Ezt követően futtathatjuk programunkat a program nevét beírva.



► Első programunk futtatása

Ha mindent jól csináltunk, fut a programunk, pontosabban, mire idáig jutunk az olvasásban, alighanem véget is ért a végrehajtása. Megírtuk az első programunkat!

Ha valamit nem írtunk be jól, akkor hibaüzenetet kapunk. Például:

```
C:\Users\raerek\programjaim>elsi.py
'elsi.py' is not recognized as an internal or external command,
operable program or batch file.
```

► Elgépeltük a fájlnevet. Ez nem programozási hiba, az operációs rendszer jelzi a hibát.

```
C:\Users\raerek\programjaim>elso.py
  File "C:\Users\raerek\programjaim\elso.py", line 1
    print('Szia.)
                 ^
SyntaxError: EOL while scanning string literal
```

► Valamit a programban írtunk rosszul. Ez a Python hibaüzenete.

Figyeljük meg, hogy miből áll egy hibaüzenet, alighanem sokat látunk még ilyet. A Python

- megmondja, hogy melyik sorban van a hiba (line 1),
- mutatja, hogy szerinte hol a hiba (néha pontatlanul),
- meg is fogalmazza a hibát (utolsó sor).

A programunkat nem kellett külön gépi kódúvá alakítanunk. Az átalakítást a Python automatikusan végzi a háttérben, mert a Python úgynevezett interpretált (értelmezett) nyelv. Megjegyezzük még, hogy programunk a parancssori felületen fut, és még jó darabig nem foglalkozunk grafikus felületű szoftverekkel, mert készítésük lényegesen bonyolultabb.

Az IDE

A szövegszerkesztővel történő programírás után más módszert, más fejlesztési környezetet használunk. Az IDE (Integrated Development Environment – integrált fejlesztői környezet) egy olyan alkalmazás, amely segít nekünk a program megírásában. A Python saját, alapértelmezett fejlesztői környezetének neve: IDLE. A név szóvicc: a szó hasonlít az IDE-re, de angolul tétlent jelent, holott mi épp itt fogunk sokat ténykedni.

Amikor az IDLE elindult, egy úgynevezett Python Shellt látunk, amiben szintén lefuttathatók a Python programok, bár mi ebben a könyvben mindig a parancssorban futtatjuk őket – ez csak egyéni ízlés kérdése.

A *File* menüből tallózva nyissuk meg az előbb elkészült programunkat, és látjuk, hogy az IDLE színesen jeleníti meg a programunkat – ebben segít nekünk az IDLE a Jegyzettömbhöz képest. Mostantól itt készítjük a programjainkat – az IDLE megnyitása után ne felejtsünk majd mindig új fájlt kérni, ne a Python Shellben akarjunk programot írni.

Szöveg és szám

Módosítsuk a programunkat úgy, hogy „Szia” helyett írja ki születésünk évét! Ez az adat egy szám, azaz nem kell aposztrófok közé tennünk. A programozási nyelvekben az a szokás, hogy a szöveges adatokat idézőjelek vagy aposztrófok közé kell tenni. A Pythonban mindkettőt használhatjuk, a megkötés az, hogy amelyiket a szöveg elejére írjuk, azt kell a végére is. Mi a könyvben a következő példakód kivételével az aposztrófoknál maradunk.

A számokat is írhatjuk idézőjelbe, ilyenkor a Python szövegként kezeli őket. Mit jelent ez? A legegyszerűbb, ha kipróbáljuk ezt a programot (mostantól számozzuk a programsorokat, úgy könnyebb beszélni róluk):

```
1. print('Hú' + "Ha')
2. print(2 + 3)
3. print("2" + '3')
```

Aposztróf és idézőjel is jó!

A számokat értelmezi és összeadja.

A szövegeket meg sem próbálja értelmezni, csak egymás mellé írja.

Változók

Írjunk új programot!

```
1. állat = 'ló'
2. print('állat')
3. print(állat)
```

Értéket adunk a változónak.

Kiírja a változó ÉRTÉKÉT.

Az első sor új elemet tartalmaz. Az idézőjelek nélküli szöveg itt egy *változó*, aminek azt adtuk értékül, hogy „ló”. Legegyszerűbb, ha a változókra olyan dobozként gondolunk, amibe bármit tehetünk – itt épp egy szöveget tettünk bele. Ezt a programunk megjegyzi, és ha legközelebb a doboz (változó) nevét írjuk le (idézőjelek nélkül), akkor behelyettesíti oda a doboz *tartalmát*. Ha a változó nevét aposztrófok közé írjuk, akkor a Python egyszerű szövegként tekint rá, amit meg sem próbál értelmezni.

A változókat azért hívjuk változóknak, mert az értékük változhat. Ha új értéket adunk nekik, a régi egyszer s mindenkorra nyomtalanul eltűnik. Gondoljuk végig az alábbi program kimenetét, aztán futtassuk a programot, hogy kiderüljön, jól tippeltünk-e.

```
1. állat = 'ló'
2. print(állat)
3. állat = 'nandu'
4. print(állat)
5. állat = 'cickány'
6. print(állat)
```

Szabály, hogy a Python változónevei

- betűvel vagy alávonással (_) kezdődhetnek;
- betűvel, számmal vagy alávonással folytatódhatnak (írásjel és szóköz nem lehet bennük), azaz `anyu kora` helyett használjuk az `anyu_kora` alakot (ez számít pythonosnak), vagy írjuk egybe a szavakat;
- a kis- és a nagybetű használatára figyelnek, azaz `Majom`, `majom` és `majoM` három külön változó;
- nem egyezhetnek meg az úgynevezett „foglalt szavakkal” – ilyen például a *for*, az *if*, vagy a *while*.

Nem szabály, de érdemes akként tekinteni rá: a programnak mindegy, hogy miként nevezzük el a változókat. Ha a fenti programban az „állat” helyett *mindenhol* „növény” szerepelne, a program hibátlanul működne. A *programozónak* fontos a jó változónév, hogy ha holnapután előveszi a programját, még mindig el tudjon igazodni rajta. A változónevek választásával a program értelmezését segítjük.

Adat bekérése a felhasználótól

A legtöbb program kér adatokat a felhasználótól. A telefonunkba be kell írni az új telefonszámot, vagy egy listából kiválasztani a már rögzítettet. A böngészőnkbe beírjuk, hogy melyik webhelyet nyissa meg. A gépünknek megadjuk a jelszavunkat.

Pythonban a felhasználótól az `input` utasítással kérhetünk adatot. Az utasítás legegyszerűbb formája az `input()`, így, két zárójellel. Írjunk be ennyit egy programba, és futtassuk le! Azt látjuk, hogy a program vár. Ha nyomkodjuk a billentyűket, akkor amit lenyomtunk, kiíródik, ha Entert nyomunk, a program futása befejeződik.

Ha a programunkat úgy módosítjuk, hogy a zárójelek között megadjuk, hogy mit kérdezünk a felhasználótól: `input('Hogy hívnak?')`, akkor ez kiíródik. A programunk azonban nem jegyzi meg, amit válaszolunk, mert nem mondtuk neki.

Így tudjuk erre „megkérni”:

```
1. név = input('Hogy hívnak?')
2. print(név)
```

Amit a felhasználó mond, azt betesszük egy változóba.

A print utasítás több dolgot is ki tud írni egymás után. Amit ki akarunk íratni, azt a zárójelen belül, vesszővel elválasztva kell felsorolnunk. Például: `print('Ezt', 'egymás mellé', 'írom.')`. Bármelyik szöveg helyett írható változó. Próbáljuk meg kiegészíteni a fenti programot úgy, hogy a Python nevünkön szólítva bennünket, köszönjön nekünk!

Változók, kiíratás, adat bekérése

Kérdések, feladatok

1. Mit írnak ki az alábbi programok? Gondoljuk végig, aztán próbáljuk ki a programokat, nézzük meg, hogy igazunk volt-e!

```
1. állat = 'ló'
2. ló = 'Ráró'
3. állat = 'macska'
4. print(állat)
```

```
1. gyümölcs = 'alma'
2. gyümölcs = 'körte'
3. alma = 'dinnye'
4. dinnye = 3
5. gyümölcs = alma
6. print(gyümölcs)
```

```
1. autó = 'Trabant'
2. autó = 'Renault'
3. autó = 3 - 5
4. print(autó)
```

2. Rajzoljuk ki az alábbi ábrát karakterek használatával!

A visszaper („\”, backslash) különleges szerepű, az utána lévő karakter úgynevezett vezérlőkarakter, és a Python értelmet tulajdonítana neki, nem kiírná. Ha azt akarjuk, hogy a visszaper kiíródjon, két visszapert kell írnunk – a `print('\\')` parancs csak egy visszapert ír ki.

3. Rajzoljuk ki az alábbi ábrákat karakterek használatával!

4. Kérdezzük meg a felhasználótól (a program használójától) a vezetéknevét! Kérdezzük meg a keresztnevét is, és köszönjünk neki, a teljes nevén szólítva!
A 3. sor teljesíti a feladatot, de picit csúnya: a felkiáltójel elkülönül az utolsó szótól, ami nem szép és nem is szabályos – mármint a helyesírás szabályai szerint. A 4. sor ezt oldja meg. A print utasítás alapértelmezés szerint szóközt tesz a vesszővel felsorolt kiírandók közé, ezt bíráljuk fölül a `sep=''` utasítással. A `sep` a separator (elválasztó) rövidítése, a két aposztrófunk között pedig semmi sincs: ne legyen elválasztó. Ha ezt beállítjuk, nekünk magunknak kell figyelnünk a szóközökre, ezért alakul át a negyedik sor többi része is.

```
1. vezeték = input('Mi az Ön becses vezetékneve?')
2. kereszt = input('Érdeklődhetek a keresztneve felől is?')
3. print('Üdvözlöm,', vezeték, kereszt, '!')
4. print('Üdvözlöm, ', vezeték, ' ', kereszt, '!', sep='')
```

Számok és karakterláncok a programunkban

Eddig még csak karakterláncot (szöveget) tároltunk a változóinkban. Ebben a leckében változtatunk ezen, és számokat is használunk.

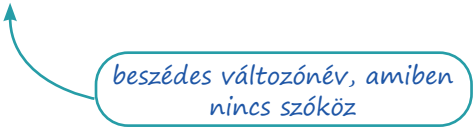
Hány éves a felhasználó?

Olyan programot fogunk írni, amely választ ad a fenti kérdésre. Az első részfeladat egy olyan program megírása, amely megkérdi, hogy mikor születtünk, és ezt ki is írja nekünk. Ez még nem tartalmaz új tudáselemet, úgyhogy készítsük el egyedül, majd olvassunk tovább!

A következő részfeladat a felhasználó korának meghatározása. Ehhez a programunknak tudnia kell, hogy melyik évben futtatják. A jelenlegi év megkérdezhető az operációs rendszertől, de egyelőre megelégszünk azzal, hogy változóként felvesszük a programunkba.

Azokat az értékeket, amelyek később nem változnak, konstansnak nevezzük, és vannak olyan programozási nyelvek, amelyeknél külön jelöljük. A Pythonban úgy *szokás*, hogy az ilyen értéket tároló változóknak csupa nagybetűs nevet adunk. A konstansnak tekintett változókat szokás a program elején megadni, azaz a programunk mostanra nagyjából így néz ki:

```
1.  IDEI_ÉV = 2021
2.  felhasználó_kora = input('Hány éves vagy?')
3.  print('Te most', felhasználó_kora, 'éves vagy.')
```



A negyedik sorunk alighanem egy kivonás lesz, például

```
születési_év = IDEI_ÉV - felhasználó_kora
```

vagy hasonló. Ha ebben az állapotban lefuttatjuk a programunkat, a kérdésre még megvárja a választ, de utána hibaüzenettel leáll.

Hányadik sorra vonatkozik a hibaüzenet?

A hibaüzenet az újonnan beírt sorban van, és átböngészve találunk benne olyat is, hogy `int` és `str`, előttük meg egy kivonásjel. Ez a három dolog a lényeg.

A Python azt igyekszik elmagyarázni, hogy nem tud egy egész számból (angolul: integer, röviden `int`) kivonni egy karakterláncot, más szóval szöveget (angolul: string, `str`).

Ha úgy gondoljuk, hogy itt valami tévedés lesz, mi rendes felhasználó módjára a programban feltett kérdésre számmal válaszoltunk, akkor igazunk van, de el kell fogadnunk, hogy a Python meg óvatos. Nem tudhatja, hogy amit válaszoltunk a kérdésére, azt biztosan számnak, tízes számrendszerbeli számnak gondoltuk. Ezért minden, amit az `input` a programnak átad, szöveg, azaz `str` marad. Ha 15-öt válaszoltunk az előző kérdésre, a program lát egy 1-es és egy 5-ös karaktert, de csak mint két egymás utáni karaktert, nem pedig egy számot.

Mik is azok a műveleti jelek?

Még iskoláskorunk legelején megtanultuk, hogy mik azok, de ha meg kell fogalmazni, esetleg elbizonytalanodunk. Végül talán arra gondolunk, hogy a műveleti jel olyan jel, ami a mellette álló adattal, adatokkal egy műveletet végez.
Futtassuk az alábbi egyszerű programot, és gondolkodjunk el a kimenetén!

```
1. print(10 + 3)
2. print('10' + '3')
3. print('Ej' + 'nye!'
4. print(10 * 3)
5. print(10 * '3')
6. print(10 * 'Abc')
```

Az aposztrófok közé írt szám NEM szám!

Fogalmazzuk meg a tanulságokat:

Az összeadásjel:

- két számot összead,
- két karakterláncot egymás mellé ír.

A szorzásjel:

- két számot összeszoroz,
- számot észlelve a karakterláncot egymás mellett a számnak megfelelően megismétli.

Az, hogy a műveleti jel pontosan milyen műveletet végez, *attól is függ, hogy az adat milyen* ***típusú****.* Ezért nem találgat a Python, hanem azt várja, hogy pontosan adjuk meg az adat típusát.

Típusátalakítás

Térjünk vissza a felhasználó születési évét kiíró programunkhoz. Azt már tudjuk, hogy az `input` utasítás karakterláncot ad vissza, és azt is, hogy nekünk számra van szükségünk. A típusátalakítást, típuskonverziót a Pythonban a céladattípus nevével megegyező utasításokkal végezzük el: az `int('2021')` utasítás eredménye 2021, számként. Ennek figyelembevételével programunk így alakul:

```
1. IDEI_ÉV = 2021
2. felhasználó_kora = input('Hány éves vagy? ')
3. print('Te most', felhasználó_kora, 'éves vagy.')
4. felhasználó_kora = int(felhasználó_kora)
5. születési_év = IDEI_ÉV - felhasználó_kora
6. print('Ekkor születtél: ', születési_év, '.', sep='')
```

„Kozmetikai” szóköz, hogy szebb legyen a kimenet.

A **típuskonverzió** a negyedik sorban van. Próbáljuk ki a kész programot!
Szeretnénk *pontosan* érteni, hogy mit csinál a típuskonverziót végző utasítássor, úgyhogy most mondjuk ki fennhangon, hogy mit csinálnak az alábbiak!

- szám = 2 + 4517
- majmok = orangutánok + cerkófok

Remélhetőleg nagyjából ezeket mondtuk:

- Adjuk össze a két számot, és az eredményt tegyük a „szám” változóba!
- Olvassuk ki az orangutánok és a cerkófok változó tartalmát, és az eredményt tegyük a „majmok” változóba!

Ha megfigyeljük a mondatainkat, látjuk, hogy előbb foglalkozunk a fenti sorok egyenlőségjeltől jobbra álló oldalával, és csak utána a bal oldallal.

Eddig három műveleti jelünk (operátorunk) volt, a „+”, a „–” és a „*” jel, de mostanra el kell fogadnunk, hogy programozáskor az egyenlőségjel is műveleti jel. Alapvetően mást jelent ilyenkor az egyenlőségjel, mint matematikaórán. Ott állításokat, kijelentéseket fogalmaztunk meg vele (Kettő egyenlő: háromból egy. Hatszor hat egyenlő harminchattal.). Programozáskor az egyenlőségjel egy művelet elvégzésére való *felszólítás*, nézzük csak meg az előbbi számos és majmos mondatot!

Programíráskor az egyenlőségjel az **értékadás** műveletének műveleti jele, olvashatjuk „legyen egyenlő” formában. Értékadáskor mindig az egyenlőségjel jobb oldalát értékeli ki a program elsőként, majd a kapott eredményt értékül adja az egyenlőségjel bal oldalán lévő változónak.

A `felhasználó_kora = int(felhasználó_kora)` sor tehát a következőt jelenti:

1. Olvassuk ki a `felhasználó_kora` változó tartalmát (ez az egyenlőségjel jobb oldalán lévő `felhasználó_kora`)!
2. A kapott értéket adjuk át az `int` utasításnak (ami majd egész számmá alakítja)!
3. Az `int` utasítás eredményét adjuk értékül a `felhasználó_kora` változónak (felülírva ezzel a régi értéket)!

Szemléletes, ha úgy képzeljük el, hogy a dobozból kivesszük a benne lévő szöveget, átgyúrjuk számmá, aztán visszatesszük ugyanabba a dobozba.

És amit nem lehet átalakítani számmá?

Abból bizony hibaüzenet lesz.
Nézzük a programunk alábbi két futtatását!

```
C:\Users\raerek\programjaim>hanyeves.py
Hány éves vagy?
Te most  éves vagy.
Traceback (most recent call last):
  File "C:\Users\raerek\programjaim\hanyeves.py", line 4, in <module>
    felhasználó_kora = int(felhasználó_kora)
ValueError: invalid literal for int() with base 10: ''

C:\Users\raerek\programjaim>hanyeves.py
Hány éves vagy? csillijómillijó
Te most csillijómillijó éves vagy.
Traceback (most recent call last):
  File "C:\Users\raerek\programjaim\hanyeves.py", line 4, in <module>
    felhasználó_kora = int(felhasználó_kora)
ValueError: invalid literal for int() with base 10: 'csillijómillijó'
```

Nem adunk meg értéket, csak Entert nyomunk.

A semmit a Python nem tudja számmá alakítani.

A szöveget sem.

Természetesen kezelhetők az ilyen jellegű hibák, csak mi még nem tanultuk meg a módját. Még sokáig abból indulunk ki programkészítéskor, hogy a felhasználótól érkező bemenetet nem kell ellenőriznünk, validálnunk. Egy „igazi” alkalmazás esetében a hibákra való felkészülés (felhasználótól kapott rossz bemenet, elfogyó háttértár, megszakadó hálózati kapcsolat stb.) a programnak igen jelentős része.

Karakterlánccá alakítás

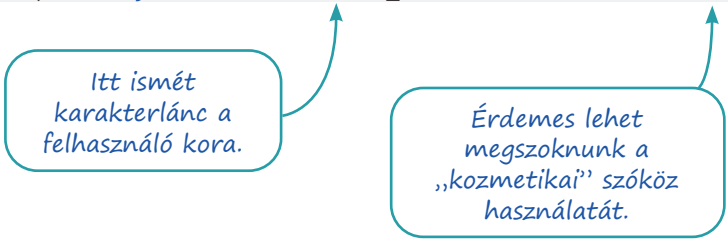
Láttuk már, hogy két karakterlánc összeadható, és azt is láttuk, hogy a `print` utasításnak több kiírnivaló is átadható, és ezeket vesszővel választjuk el. De lehet olyat írni, hogy `print('alma'+ ' ' + 'körte' + ' ' + 'dió')`? Hogyne! Ilyenkor előbb „összeadódnak” a karakterláncok, és ezt követően egyetlen karakterláncot kap meg a `print`. Persze itt éppen megvagyunk e módszer nélkül, mert a vesszővel való felsorolás remekül működik (lásd a programunk 6. sorát), de van, ahol ez probléma.

Egészítsük ki a programunkat: kérdezze meg, hogy „És milyen N évesnek lenni?”, ahol N természetesen a felhasználó korábbi válaszában szereplő szám!

Logikusnak tűnik egy `ilyen = input('És milyen', felhasználó_kora, 'évesnek lenni?')` megoldás, de sajnos az `input` nem ismeri a `print` vesszős összefűzési módszerét. Ha a „+” operátort használjuk, akkor egy másféle hibába csöppenünk, de azért próbáljuk csak ki, könnyű lesz korrigálni!

A hibaüzenet ismét `int`-ről és `str`-ről beszél, és ekkor már gyanítjuk, hogy az lehet a baj, hogy a `felhasználó_kora` a 4. sor óta egész szám, amit nem tud a Python „összeadni” egy karakterlánccal. Amikor egész számmá akartunk alakítani, az `int` utasítást használtuk. Amikor karakterlánccá alakítunk, az `str` utasításra van szükség. Programunk utolsó két sora ezt a formát ölti:

```
7. felhasználó_kora = str(felhasználó_kora)
8. ilyen = input('És milyen ' + felhasználó_kora + ' évesnek lenni? ')
```



Számok és karakterláncok

Feladatok

1. Mit írnak ki az alábbi programok?

```
1. szám = 4
2. szám = szám + 2
3. print(szám)
```

```
1. edény = 'bögre'
2. edény = 'sárga' + edény + 'görbe' + edény
3. print(edény)
```

Természetesen tudunk tizedestörtekkel is dolgozni programjainkban. A tizedestörteket a programozók lebegőpontos számnak is nevezik, ami angolul *floating point number*, innen származik a típus neve: `float`. A megfelelő típuskonverziót a `float` parancs hajtja végre (használni úgy kell, mint az `int` és az `str` utasítást). A tizedesvessző helyett tizedespontot használunk.

2. Kérjünk be egy kilométerben mért távolságadatot a felhasználótól, és írjuk ki tengeri mérföldre átváltva! (Egy tengeri mérföld 1852 méter.)
   Ha elkészültünk, megírhatjuk a feladat megfordítását.
3. Kérjünk be a felhasználótól két számot, tároljuk őket egy-egy változóban!
   a. Adjuk össze őket, és írjuk ki az eredményt!
   b. Írjuk ki az eredmény elé, hogy „Az összegük:”!
   c. Írjuk ki magukat a számokat is! Ha például 2-t és 3-at adott meg a felhasználó, akkor a kimenet legyen ilyen:
   `2 és 3 összege: 5.`
4. Írjuk át a programot szorzásra! (A szorzás jele a *, az osztásé a /, a hatványozásé a **. A gyökvonásnak nincs külön jele, de tört kitevővel megoldható.)
5. Vegyük elő az előző gyakorló feladatsor 4. feladatának megoldását. Ebben a feladatban már megoldottuk a vezeték- és a keresztnév kiírását. Bővítsük úgy a programot, hogy kérdezze meg a születési évünket is, és írja ki a nevünkkel egy sorban: `Kék Blamázs, 2011.` A születésiév-megállapító programunkkal egybegyúrva készíthetünk olyan programot is, amelyik megkérdezi a neveinket, a születési évünket, és a bulvárcikkekben szokásos formában, a korunkkal együtt írja ki a nevünket: `Fehér Karnis (21)`.
6. Adott óra, perc, másodperc hármassal megadott időt váltsunk át másodpercre! (Az adatokat nem kell mindenképp a felhasználótól kérni, beírhatjuk őket a programba is.) Sikeres megoldás után készítsük el a feladat megfordítottját!
7. Kihívást jelentő feladat: Milyen szöget zár be egymással a kis és a nagy mutató adott időpontban?

Az egyik utasításnak közvetlenül is átadhatjuk a másiktól visszakapott értéket, így nem kell külön sorba írnunk a típuskonverziót:

8. Milyen típusú adat lesz a „szám” változóban? Lefut-e rendben a második sor?

```
1. szám = int(input('Hányat ugrik a nyuszi? '))
2. print('A nyuszi ' + str(szám) + '-at ugrik.')? ')
```

Elágazások

Eddig csupa olyan programot írtunk, ami elkezdődött az elején, sorban egymás után végrehajtott minden utasítást, aztán kilépett. Ebben a leckében ezen változtatni fogunk.

Gondolunk egy számra

A témakör első leckéjében már láttunk egy olyan programot, amelyikben elágazás van. Az a program mást csinál, ha nem vagyunk még tizennégy évesek, és mást, ha már betöltöttük ezt az életkort. Hasonló működésű az ugyanott megismert „Mi a neve Mátyás királynak?” program.

Az alábbi **folyamatábra** is egy hasonló programot ír le. Nincsenek benne konkrét utasítások, mert fontosabb, hogy gondold először át, hogy mit csinál a programod, és ráérsz utána azon elmélkedni, hogy miként **kódolod** a programodat.

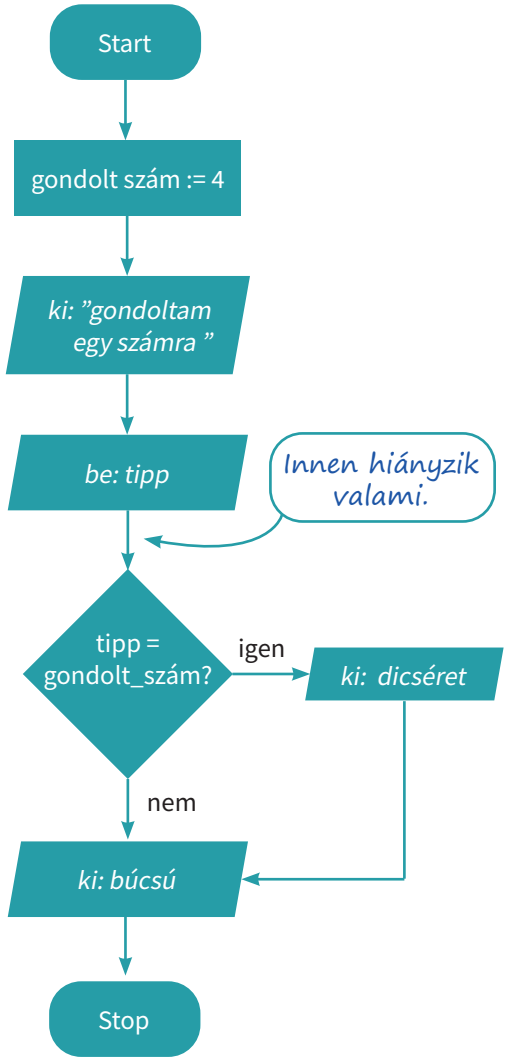
1. Mit csinál a program?
2. Mi az a lépés, amit nem tüntettünk fel? (Ha most nem jövünk rá, nem probléma: hamarosan úgyis megírjuk a programkódot, akkor szólni fog a Python.)
3. Hogyan olvassuk ki a := jelet? Mi a megfelelője Pythonban?

Miközben a programunkat folyamatábrával megfogalmazzuk, **algoritmust** (receptet) adunk a bennünket érdeklő probléma megoldására. A folyamatábrák elég látványosak, de hamar lelógnának a könyvlapról, így aztán a gyakorlatban gyakrabban használunk egy másik algoritmusleíró eszközt, a **mondatszerű leírást.**

A leírás szabályaira rá fogunk érezni.

4. Vessük össze ezt a leírást a folyamatábrával!
5. Melyik az a művelet, ami a mondatszerű leírásban már megvan, de a folyamatábrában még hiányzik?

```
gondolt_szám := 4
ki: „gondoltam egy számra”
be: tipp
tipp átalakítása egésszé
elágazás
ha tipp = gondolt_szám:
      ki: dicséret
elágazás vége
ki: búcsú
```



Készítsük el a fenti két algoritmusleíró eszközzel megadott programkódot!

```
1. gondolt_szám = 4
2. tipp = input('Gondoltam egy számra. Tippeld meg! ')
3. tipp = int(tipp)
4. if tipp == gondolt_szám:
5.     print('Ügyes!')
6. print('Pápá.')
```

A kettőspont hatására a következő sor bentebb kezdődik.

Ez tényleg 2 egyenlőségjel.

Egy TAB vagy 4 szóköz KELL!!!

Teszteljük a programunkat: adjuk meg a helyes megoldást, de próbáljuk ki helytelennel is!

A program következő változatában, más szóval verziójában kicsit bővebben dicsérünk, illetve a hibásan tippelő felhasználókat kicsit ugratjuk. A mondatszerű leírás a következő:

6. Módosítsuk ez alapján a folyamatábrát (segítség: a rombuszból lefelé nem lesz nyíl, de balra igen, és a két nyíl a rombusz alatt összetalálkozik)!

```
gondolt_szám := 4
ki: „gondoltam egy számra”
be: tipp
tipp átalakítása egésszé
elágazás
ha tipp = gondolt_szám:
      ki: kétsoros dicséret
különben:
      ki: ugratás
elágazás vége
ki: búcsú
```

A dicséret második sora újabb print utasítást jelent az előző alatt. Írjuk be az új sort:

```
print('Gratulálok.')
```

behúzva és behúzás nélkül! Mindkét esetben teszteljük a programunkat, és vessük össze a viselkedését. Fogalmazzuk meg a behúzás szerepét!

Ha megvagyunk, írjuk meg az ugratós részt is. A „különben” szó angolul „else” – ezt kell használnunk kódoláskor. A kész program:

Ez itt az elágazás FELTÉTELE.

```
1. gondolt_szám = 4
2. tipp = input('Gondoltam egy számra. Tippeld meg! ')
3. tipp = int(tipp)
4. if tipp == gondolt_szám:
5.     print('Ügyes!')
6.     print('Gratulálok.')
7. else:
8.     print('Hosszan gondolkodtál rajta?:)')
9.     print('Nem érte meg.;)')
10. print('Pápá.'))
```

Ez a két utasítás a ,,ha''-ág.

Ez a két utasítás pedig a ,,különben''-ág.

A programunk tanulságai:

1. Ami az `if` után következik, az az elágazás feltétele.
2. A feltételvizsgálatban két egyenlőségjel kell. A programozásban az egy egyenlőségjel egy felszólítás (ismerjük már, értékadáskor használjuk), a két egyenlőségjel kérdés: A tipp egyenlő a gondolt számmal?
3. Ami az elágazás ágaiban van (a fenti program 5–6. és 8–9. sora), az bentebb kezdődik. A Python onnan tudja, hogy mikor kezdődik egy ág, hogy a programsor bentebb kezdődik, és onnan tudja, hogy hol van vége az ágnak, hogy az újabb programsor már nem kezdődik bentebb.

Összetett feltétel

Szeretnénk úgy bővíteni a programunkat, hogy ha csak eggyel tippelt mellé a felhasználó, akkor ezt eláruljuk neki. Elsőként az algoritmus mondatszerű leírását módosítjuk. Az új, „különbenha” ágat megvalósító Python-utasítás az `elif`.

Akár neki is foghatnánk a kódolásnak, de hogy programozandó a „csak egyet tévedett”? Ilyen utasítás nincs, ezért az algoritmusunkat egy-két lépéssel tovább kell finomítanunk. Észrevesszük, hogy kétféleképp lehet egyet tévedni: vagy eggyel nagyobbat, vagy eggyel kisebbet tippelve. Az „eggyel nagyobbat tippelt” így írható le:

```
tipp = gondolt_szám+1.
```

```
gondolt_szám := 4
ki: „gondoltam egy számra”
be: tipp
tipp átalakítása egésszé
elágazás
ha tipp = gondolt_szám:
      ki: kétsoros dicséret
különbenha csak egyet
tévedett:
      ki: csak egyet tévedtél
különben:
      ki: ugratás
elágazás vége
ki: búcsú
```

A másik feltétel megfogalmazása nem okozhat gondot, de ezek szerint két feltétel lett az egyből. Megírhatjuk úgy az algoritmust (és a programot), hogy két „különbenha” ágat adunk meg, ugyanazzal a kiírandó üzenettel, de ez nem szerencsés – például azért, mert ha módosítani kell az üzenetet, két helyen is módosítanunk kell, és az egyiket előbb-utóbb elfelejtjük megcsinálni.

Alakítsuk inkább a két feltételünket egy összetett feltétellé:

```
különbenha tipp = gondolt_szám+1 vagy tipp = gondolt_szám-1:
```

A két feltételből így lett egy. Lévén a „vagy” szó kapcsolja őket össze, elég, ha az egyik teljesül. Ha „és” kapcsolná őket össze, mindkettőnek teljesülnie kellene, hogy a teljes összetett feltétel igaz legyen. A kód most így néz ki:

```
1. gondolt_szám = 4
2. tipp = input('Gondoltam egy számra. Tippeld meg! ')
3. tipp = int(tipp)
4. if tipp == gondolt_szám:
5.     print('Ügyes!')
6.     print('Gratulálok.')
7. elif tipp == gondolt_szám + 1 or tipp == gondolt_szám - 1:
8.     print('Ó' csak eggyel tévedtél.')
9. else:
10.    print('Hosszan gondolkodtál rajta?:)')
11.    print('Nem érte meg.;)')
12. print('Pápá.')
```

Kérdések

1. Hány `elif`-ág és hány `else`-ág szerepelhet egy elágazásban?
2. Melyiket kell utolsóként megadni?
3. Melyiknek nincs feltétele?
4. Létezhet olyan elágazás, amelyikben nincs egyik sem?
5. Kihívást jelentő feladat: Az alábbi sor nem jó (bár a program nem jelez hibát): `elif tipp == gondolt_szám + 1 or gondolt_szám – 1:` Miért nem jó?

Véletlenszám-előállítás

Meglehetősen unalmas lehet, hogy a programunk mindig a négyre gondol. A legtöbb programozási nyelvben van valamilyen módszer véletlen számok előállítására. A Pythonban több is van, ezek közül a `random.randint` (randint: random integer, azaz véletlen egész) az, amelyik véletlenszerű egész számot állít elő, más szóval generál. Ha ezt az utasítást használni akarjuk, akkor előbb be kell töltetni a programmal a `random` nevű modult. A programunk első sorai a következőképp alakulnak:

```
1. import random
2.
3. gondolt_szám = random.randint(1,6)
4. print('Súgok:', gondolt_szám)
5. tipp = input('Gondoltam egy számra. Tippeld meg! ')
```

Itt töltjük be a „random” modult. Az importálásokat követően szokás egy sort kihagyni.

1 és 6 közötti egész számot állítunk elő.

A súgás a program tesztelésekor hasznos, a végső változatból vegyük ki!

Feladatok

Az „==” operátor nemcsak számok, hanem szövegek egyezésének vizsgálatára is használható.

1. Kérjünk be jelszót a felhasználótól, és hasonlítsuk össze a programban tárolttal! Ha a felhasználó eltalálta a jelszót, írjuk ki, hogy „Helyes jelszó.”, különben „Hozzáférés megtagadva.”.

A feltételek megfogalmazásakor használható a „>”, a „<”, a „>=” és a „<=” operátor is. Azt, hogy „nem egyenlő”, a „!=” operátorral fejezzük ki, a matematikában megszokott áthúzott egyenlőségjelünk nincs.

2. Kérjünk be két számot a felhasználótól! Írjuk ki a nagyobbat!
3. Állítsunk elő két véletlen számot, és kérdezzük meg a felhasználótól az összegüket! Ha helyesen válaszol, dicsérjük meg!

```
1.  import random
2.
3.  egyik = random.randint(1,10)
4.  másik = random.randint(1,10)
5.  egyik_szövegként = str(egyik)
6.  másik_szövegként = str(másik)
7.  tipp = input('Mennyi ' + egyik_szövegként + ' és ' + másik_szöveg-
        ként + ' összege? ')
8.  tipp = int(tipp)
9.  összeg = egyik + másik
10. if tipp == összeg:
11.     print('Valóban annyi, ez igen!')
```

4. Írjunk olyan programot, amelyik bekér két csapatnevet és két pontszámot, majd kiírja a mérkőzés eredményét (a felhasználó válaszai vastagabbal szedve):

```
Mi az egyik csapat neve? Tóparti királyok
Hány pontot szerzett? 78
Mi a másik csapat neve? Talpasi csodatévők
Hány pontot szerzett? 54
Az összecsapás eredménye:
Tóparti királyok - Talpasi csodatévők
78 : 54
Tóparti királyok nyert.
```

5. Vegyük elő azt a programunkat, amelyik a felhasználó nevét és korát kezeli. A felhasználónak javasoljunk életkorának megfelelő olvasnivalót!
   a. 0–3 év: „Totyogóknak a kettes számrendszerről”
   b. 4–6 év: „Hackeljük meg az óvodát!”
   c. 7–14 év: „Felhőtechnológia a menzán”
   d. 15–18 év: „Big data a középiskolában”

6. Ha bonyolultabb összetett feltételeket fogalmazunk meg, érdemes lehet zárójelek használatával egyértelműsíteni a szándékunkat. Melyik feltételmegfogalmazás helyes az alábbiak közül?
   a. Ha (van tollunk és van ceruzánk) vagy van egy papírlapunk:
      tudunk rajzolni valamit.
   b. Ha (van tollunk vagy van ceruzánk) és van egy papírlapunk:
      tudunk rajzolni valamit.
   c. Ha van tollunk vagy (van ceruzánk és van egy papírlapunk):
      tudunk rajzolni valamit.
7. Kihívást jelentő feladat: A kistesód szülinapi banános nyaflatyát csinálod. A kistesód szerint a banános nyaflaty akkor jó, ha:
   a. nincs benne egyszerre vaníliás cukor és tortareszelék,
   b. ha van benne fahéj, akkor kell rá tejszínhab is,
   c. ha nincs benne fahéj, nem kerülhet rá tejszínhab sem.

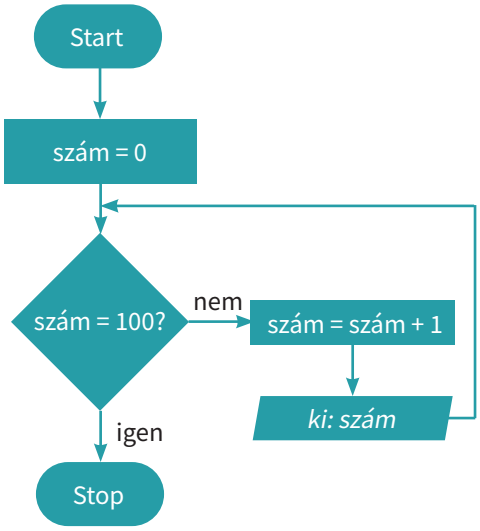
   Írj programot a nyaflaty jóságának eldöntésére!

Ciklusok

A ciklus eredetileg valamilyen ismétlődő dolgot jelent, gondolhatunk holdciklusra, választási ciklusra, árapályciklusra. Mi ebben a könyvben egy olyan *programrészletet* értünk rajta, amely valahányszor megismétlődik. Sok elterjedt programozási nyelvben a ciklus lehet számlálós vagy feltételes, de a Pythonban e kettő közül csak az utóbbi létezik, cserébe van még bejárós ciklusunk.

A feltételes ciklus (while-ciklus)

Elsőként megvizsgáljuk ezt a folyamatábrát. Vajon mit csinál a program?



Mondatszerű leírással így néz ki az algoritmusunk:

```
szám := 0
ciklus amíg szám != 100:
   szám = szám + 1
   ki: szám
ciklus vége
```

A != azt jelenti, hogy: „NEM egyenlő”

Python nyelven pedig az alábbi formát ölti programunk:

```
1. szám = 0
2. while szám != 100:
3.     szám = szám + 1
4.     print(szám)
```

A „while” annyit tesz: amíg.

„Amíg” a feltétel teljesül, addig újra meg újra belépünk a ciklusba.

Behúzás, mint az if-nél.

Ez a két sor „benne van a ciklusban” – ők a CIKLUSMAG.

1. Mi történik, ha a ciklusmag első sorát elhagyjuk? (Ha sikerült végtelen ciklusba kerülnünk, a program a Ctrl + C [C jelentheti azt, hogy Cancel, jelentése töröl, érvénytelenít] billentyűkombinációval megállítható.)
2. A != helyett milyen feltétellel érhetjük el ugyanezt az eredményt?
3. Hogyan változik a program kimenete, ha a ciklusmag két sorát felcseréljük? Ha a felcserélt sorokkal is szeretnénk a számokat 1-től 100-ig kiírni, mit kell még módosítani a programon?
4. Hogyan íratható ki minden második szám 100-ig? Hogyan íratható ki minden harmadik szám 100-ig?

Következő órára leírod százszor, hogy…

Bizonyára sokaknak viccekből, régi történetekből ismerős az alcímben jelzett tanári büntetés. Jelen sorok írójának azonban még a valóság volt: le kellett írnia százszor, hogy „A tornaterembe sapkában megyek át”. A fenti program egyetlen sorának módosításával megoldható ez a feladat, mi most mégis három módosítást is teszünk.

```
1. számláló = 0
2. while számláló != 100:
3.     számláló += 1
4.     print('Tudom, sapka.')
```

A változót átnevezzük a szerepének megfelelően. Amikor a változó értékét nem igazán használjuk a ciklusban, Pythonban szokás alávonásnak (,,_'') elnevezni a változót. Próbáld ki!

Ez ugyanaz, mint a számláló = számláló + 1 Használd a neked jobban tetszőt!

Túltenni Gausson

A kis Gauss, amikor még nem volt nagy matematikus, az anekdota szerint egész osztályával együtt azt a feladatot kapta a tanárától, hogy adja össze a számokat egytől százig. A tanár közben nekiállt valami más munkának, de a kis Gauss két perc múlva szólt, hogy készen van, és az eredmény 5050. Rájött, hogy 1 + 100 = 101, 2 + 99 = 101, és így tovább. 50-szer 101 pedig 5050. Ha már van számítógépünk, illendő a két percnél jobb eredményt hoznunk, méghozzá Gauss felismerésének kihasználása nélkül.

```
1. számláló = 0
2. összeg = 0
3. while számláló < 100:
4.     számláló += 1
5.     összeg = összeg + számláló
6. print('Összesen:', összeg)
```

Ezúttal két változó is kelleni fog, mind a kettőnek kell adnunk kezdeti értéket.

Figyeljünk a ciklus feltételének változására!

Ez már nincs behúzva: nincs a ciklusmagban.

A logikai adattípus

Vegyük elő a számkitalálós programunkat, és csupaszítsuk le annyira, hogy csak a jó válaszra reagáljon! Tervezzük át úgy, hogy kérdezzen addig, amíg ki nem találjuk a számot! A mondatszerű leírásba nem írjuk bele az importálást végző utasítást:

```
gondolt_szám := 1 és 6 közötti véletlen szám
kitalálta = hamis
ciklus amíg ki nem találta:
      be: tipp
      ha tipp = gondolt_szám:
            kitalálta = igaz
ciklus vége
```

Látjuk, hogy bevezetünk egy változót, amiben igaz vagy hamis érték tárolható. A változó kezdeti értéke hamis, és akkor állítjuk át igazra, ha a tipp jó volt. Minthogy a változót hamis értékkel inicializáltuk, a ciklusba legalább egyszer belépünk, és pont ezt akarjuk.

Azok a változók, amelyek igaz (True) és hamis (False) értéket vehetnek fel, úgynevezett logikai típusúak (a Pythonban a típus neve `bool`, George Boole matematikus után, aki efféle problémákkal való foglalatosságáról vált híressé).

Mindez kódként így néz ki:

```
1. import random
2.
3. gondolt_szám = random.randint(1,6)
4. kitalálta = False
5. while not kitalálta:
6.     tipp = int(input('Szerinted? '))
7.     if tipp == gondolt_szám:
8.         kitalálta = True
```

False és True, nagy kezdőbetűvel!

Lehetne while kitalálta == False: de így jobban olvasható a kód, pythonosabb.

Mélyebb behúzás az if miatt.

Összetett ciklusfeltétel

A program türelmes, és így persze a felhasználó a végtelenségig próbálkozhat. Írjuk át a programunkat úgy, hogy csak három próbálkozást engedjen! Számolnunk kell a próbálkozásokat, de nem elég, ha a ciklus feltételeként csak azt adjuk meg, hogy még nem használtuk el mind a három próbálkozást. Arra is figyelnünk kell, ha közben a felhasználó kitalálta a gondolt számot.

Szerencsére a `while`, pont ugyanúgy, mint az `if`, képes összetett feltételek kezelésére.

5. Hogyan tartjuk nyilván az elhasznált lehetőségeket?
6. Fogalmazzuk meg a `while` feltételét!

A teljes kód a következő:

```
1. import random
2.
3. gondolt_szám = random.randint(1,6)
4. kitalálta = False
5. számláló = 0
6. while not kitalálta and számláló < 3:
7.     tipp = int(input('Szerinted? '))
8.     számláló += 1
9.     if tipp == gondolt_szám:
10.        kitalálta = True
```

7. Helyezzünk el ismét olyan programsorokat a kódban, amelyek a felhasználó dicséretéért, ugratásáért felelnek! Több helyre is elhelyezhetőek, és mindnek megvannak az előnyei és hátrányai. Hasonlítsunk össze néhány lehetséges megoldást!
8. Szeretnénk megoldani, hogy ha a felhasználó a harmadik lehetőségre már majdnem kitalálta a megoldást (csak egyet tévedett), akkor kapjon még egy esélyt. Ahogy programozáskor mindig, ismét több helyes megoldás lehetséges – valósítsuk meg a nekünk legjobban tetszőt!

Ciklusok és véletlenek

Feladatok

1. Ciklussal meg tudunk oldani egyenleteket az egész számok halmazán – próbálgatással.
   a. Oldjuk meg a $3x + 2 = 59$ egyenletet a pozitív egészek halmazán!

```
1. x=1
2. while 3*x + 2 != 59:
3.     x = x + 1
4. print('A megoldás:', x)
```

   b. Oldjuk meg a $6x^2 + 3x + 8 = 767$ egyenletet az egész számok halmazán! Honnan érdemes indítani a próbálgatást?
   c. Diophantosz ókori görög matematikus verses sírfelirata több fordításban fellelhető az interneten. A felirat alapján fogalmazz meg egyenletet, és írj programot, ami megoldja! Hány évesen halt meg Diophantosz?
   d. Ilyen módszerrel csak egész gyökű egyenlet oldható meg biztosan. Miért?
   e. Ha az egyenletnek nincs egész gyöke (például $6x^2 + 3x + 8 = 768$), akkor végtelen ciklusba kerül a programunk. Miként biztosítható, hogy ne lépjen végtelen ciklusba a program, és ha már esélytelen, hogy talál megoldást, ne próbálkozzon tovább? Gondolkozzunk összetett feltételben!
2. Írjunk pénzfeldobás-szimulátort!
   a. A gép írja ki, hogy fejet vagy írást „dobott”! Használhatjuk a `random.randint` utasítást is, de van más megoldás is.

```
1. import random
2.
3. dobás = random.randint(1,2)
4. if dobás == 1:
5.     print('fej')
6. else:
7.     print('írás')
```

```
1. import random
2.
3. dobás = random.choice(['fej', 'írás'])
4. print(dobás)
```

   b. Készítsünk statisztikát! Egymillió feldobásból mennyi lesz fej, és mennyi írás?
   c. Írjuk át a programot úgy, hogy kockadobásokat számoljon!
   d. Készítsünk cinkelt kockát! A hatos jöjjön ki kétszer akkora eséllyel, mint a többi szám!
3. Írjunk randiszimulátort!
   a. A számítógép kérdezze azt, hogy „Szeretsz?”, amíg azt nem válaszoljuk, hogy „Nagyon!”, vagy azt, hogy „Jobban, mint a kókuszgolyót!”.
   b. A számítógép legyen durcás: legfeljebb három kérdés után zavarjon el bennünket, ha nem adjuk meg a „helyes” válaszok valamelyikét!
   c. A számítógép legyen szeszélyes: zavarjon el bennünket 2-4 „rossz” válasz után véletlenszerűen!

Ciklusok oda-vissza és egymásba ágyazva

Feladatok

1. Írjuk ki a számokat csökkenő sorrendben 100 és –100 között!

A `print` utasítás mindig új sort kezd a kiírnivalót követően. Van ugyanis egy `end` nevű paramétere, ami alapértelmezetten, azaz ha mást nem adunk meg: `'\n'`. Egy visszaper és egy `n` betű. A visszaper (backslash) szól, hogy a következő betű nem is betű, hanem vezérlőkarakter, az `n` pedig azt jelenti: new line, új sor. Ezt azonban felülbírálhatjuk.

2. Írjunk ki egy számsort, aztán vizsgálódjunk még egy keveset!
   a. Írjuk ki (két) ciklussal, hogy „12345678987654321”!

```
1. szám = 1
2. while szám < 9:
3.     print(szám, end='')
4.     szám += 1
5. while szám > 0:
6.     print(szám, end='')
7.     szám -= 1
8. print('')
```

A sor végén itt nem '\n' van, és nem is szóköz, hanem egy nagy semmi. Azaz: NE KEZDJ ÚJ SORT!

Itt viszont a nagy semmit írjuk ki, aminek a végén ott van a '\n' – azaz az utoljára kiírt egyes után kérünk egy sortörést.

   b. Hogyan változik a programunk, ha elhagyjuk az utolsó sort?
   c. Hány helyen kell módosítanunk a programot, hogy ne kilencig, hanem kilencmillió-kilencszázkilencvenkilencezer-kilencszázkilencvenkilencig írja a számokat?
   d. Hogyan változik a program futásának sebessége, ha nem íratjuk ki a számokat, csak elszámoltatunk oda-vissza? (Írjunk a program harmadik és hatodik sorának az elejére kettős keresztet (#)! Így ezeket a sorokat megjegyzéssé alakítottuk, nem hajtódnak végre.)
3. Írjunk egymás mellé 10 csillagot („*”) úgy, hogy a programkódban csak egyetlen csillag karakter legyen!
4. Csillagok több sorban
   a. Írjunk egymás alá öt csillagsort! Ötlet: tegyük az előző feladat ciklusát egy másik ciklus belsejébe, figyelve a behúzásokra!

```
1. sorszámláló = 1
2. while sorszámláló <= 5:
3.     csillagszámláló = 1
4.     while csillagszámláló <= 10:
5.         print('*', end='')
6.         csillagszámláló += 1
7.     print('')
8.     sorszámláló += 1
```

A két ciklusnak két külön számlálója van.

Minden sor elején újrakezdjük a csillagok számolását.

Ez a három programsor ír ki egy sornyi csillagot.

Minden sornyi csillag után lezárjuk a sort, és újat kezdünk.

A belső ciklus magja nagyobb behúzást kap.

b. Az előző feladat megoldásának egyetlen helyen való megváltoztatásával alakítsuk háromszöggé a program kimenetét! (Ötlet: egy sorban annyi csillagot kell kiírni, ahányadik…)

```
*
**
***
****
*****
```

c. Ha elkészültünk, írjuk át úgy a programot, hogy minden sorban csak az utolsó csillag jelenjen meg!
d. Minden sorban az első és az utolsó csillag jelenjen meg!

5. Írjunk szorzótáblát a kicsiknek! A várt kimenet:

```
1 * 1 = 1
1 * 2 = 2
…
6 * 6 = 36
6 * 7 = 42
…
10 * 9 = 90
10 * 10 = 100
```

Összetartozó adatok kezelése

A lista adattípus

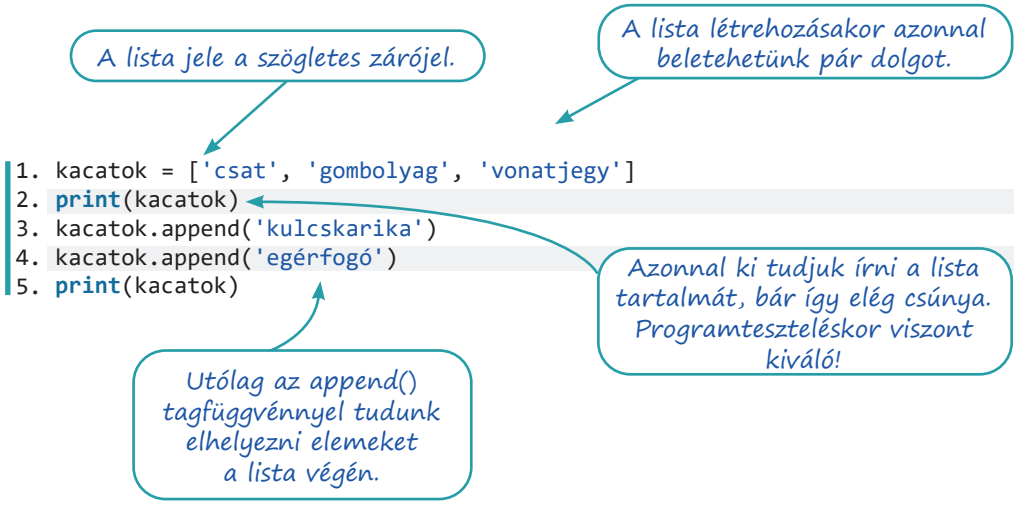
A programjainkban az adatokat változókban tároljuk, és eddig négy változótípust, adattípust ismerünk:

- a karakterláncot (szöveg, string, `str`),
- az egész számokat (integer, `int`),
- a valós, tizedestörtként megjelenő számokat (lebegőpontos szám, `float`)
- és a logikai értékeket (`True` és `False`, azaz a `bool` típus).

Ezek közül egyik sem jó akkor, amikor több, egymáshoz tartozó adatot szeretnénk tárolni, kiírni, műveletet végezni velük. Például tárolni szeretnénk az osztályunkba járók neveit. Beleírhatjuk egyetlen karakterláncba őket, de aztán hogyan írjuk ki őket egyesével? Vagy ábécérendben? Miként tároljuk az osztálylétszámokat, a kontinensek méreteit, az ország településeinek lakosságát, azt, hogy a majonézek hány grammosak és mennyibe kerülnek, a kedvenc sorozatunk szereplőit, a szomszéd kiskutyáinak neveit?

Mindezt megoldja egy **összetett adattípus** használata. Ilyen például a Python **lista** adattípusa.

Tároljuk kacatjainkat listában!



A Python listája megtartja az adatok sorrendjét, azaz mindig 'csat', 'gombolyag', 'vonatjegy' sorrendben kapjuk vissza az adatokat és nem másképp.

Nem kell egyforma típusú adatokat tenni egy listába, azaz teljesen jó a

```
csata = ['Isaszeg', 1849, 'április', 6, 'magyarok']
```

listamegadás, egy listában a csata helyszínével (szövegként), évével (egészként), hónapnevével (szövegként) és napjával (egészként), valamint a győztes oldallal (szövegként).

A listaelemek sorszámozása

Az adatok sorszámot is kapnak, mégpedig nullától kezdődően. A nullától való számozás a számítógépek világában nem szokatlan. Szokjuk meg mi is, hogy az előző listának öt eleme van, de az utolsó a negyedik sorszámú. Így az előző lista nulladik eleme `'Isaszeg'`, a harmadik pedig 6. A sorszámra a programunkban így hivatkozunk:

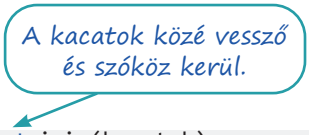
- a lista nulladik eleme: `csata[0]`
- a lista második eleme: `csata[2]`
- a lista második és az utáni elemei: `csata[2:]`
- a lista másodikat *megelőző* elemei: `csata[:2]`
- a lista másodiktól a negyediket *megelőzőig* terjedő (tehát a második és a harmadik) elemei: `csata[2:4]`
- a lista utolsó eleme: `csata[-1]`

Akiben felmerül a kérdés, hogy „És hogyan tárolnám a tavaszi hadjárat összes csatáját egyetlen listában?”, azt megnyugtatjuk, hogy lehet egy lista eleme egy másiknak. Aki ettől megijed, azt is megnyugtatjuk: ebben a könyvben nem foglalkozunk ilyen listákkal.

Listák kiírása

Láttuk már, hogy egy lista egyszerűen kiírható a `print(listanév)` utasítással. Láttuk azt is, hogy így nem szép, kell ennél valami jobbnak lennie. A megoldást egy `join()` nevű tagfüggvény jelenti, amelyiknek a használata elsőre talán meglepő. A zárójelben átadott lista elemeit fűzi össze egyetlen karakterlánccá, az elemek közé pedig az elején, aposztrófok között megadott karakterláncot teszi.

Egészítsük ki az előző programunkat az alábbi sorokkal!




```
6. kacatok_felsorolva = ', '.join(kacatok)
7. print('A kacatjaim: ', kacatok_felsorolva, '.', sep='')
```

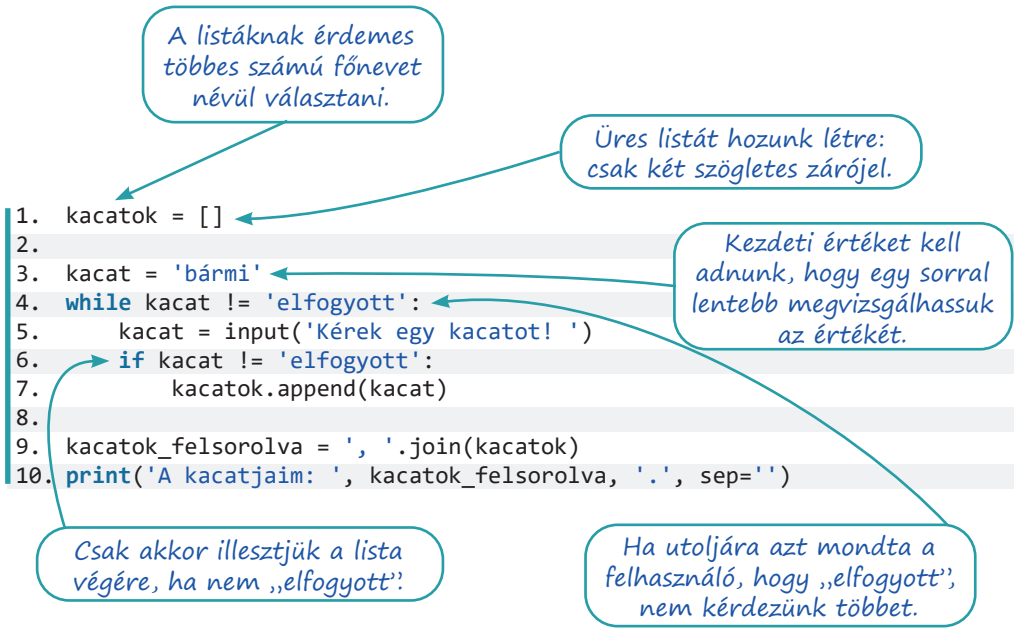
A `join()` csak karakterláncokat tud összefűzni, azaz majd ügyeskednünk kell, ha egy számokból álló listát kell kiírnunk vele. A fenti „csata” listával (amiben vegyesen vannak karakterláncok és számok) végképp nehezen boldogul.

Lista feltöltése a felhasználó által megadott adatokkal

A kacatos listát a felhasználó saját kacatjaival töltjük fel. Minthogy senkinek sincs csak egy kacatja, ciklust szervezünk a feladatra. A kacatokat egyesével kérdezzük meg, és mindegyiket hozzáfűzzük a bővülő lista végére. De honnan fogjuk tudni, hogy befejezhetjük, nincs több kacat?

Meg kell vizsgálnunk minden kacatot, még mielőtt beillesztjük a lista végére, és ha például a kacat neve az, hogy „elfogyott”, akkor befejezzük a lista feltöltését, és kiírjuk, amit kaptunk.

Írjuk meg a programunk első változatát, és próbáljuk ki!




```
1.  kacatok = []
2.
3.  kacat = 'bármi'
4.  while kacat != 'elfogyott':
5.      kacat = input('Kérek egy kacatot! ')
6.      if kacat != 'elfogyott':
7.          kacatok.append(kacat)
8.
9.  kacatok_felsorolva = ', '.join(kacatok)
10. print('A kacatjaim: ', kacatok_felsorolva, '.', sep='')
```

A harmadik sorban csak azért adjuk a „bármi”-t a kacat változó értékéül, hogy egy sorral lentebb megnézhessük, hogy az érték nem „elfogyott”-e. A „bármi” helyett akármi más is szerepelhet itt, csak az „elfogyott” nem, mert akkor egyszer sem lépnék be a ciklusba. A „bármi” értéket nem használjuk sehol, viszont zavaró lehet. A Python az ilyen esetekre tartogatja a nagybetűs semmit, azaz a `None` értéket. Ha a harmadik sort így írjuk át:

```
kacat = None
```

akkor lényegében azt mondjuk a Pythonnak: „Kérünk egy kacat nevű változót, de még ne tegyünk bele értéket.” A negyedik sorban a `while` tudni fogja, hogy a `None` nem ugyanaz, mint az „elfogyott”, és belép a ciklusba.

Informatikusszemmel nézve nem túl szerencsés, amit a hatodik (és később a negyedik) sorban művelünk. Itt ugyanis onnan tudjuk, hogy nincs több kacat, hogy egy különleges nevű kacatot adunk meg. Ha egyszer véletlenül a felhasználónak mégis lesz „elfogyott” nevű kacatja, nem tudja bevinni a programunkba.

Átírható úgy a program, hogy az `'elfogyott'` helyett az üres karakterláncot tekintse a felsorolás végének, azaz az `'elfogyott'` helyett a negyedik és a hatodik sorban is használhatunk két aposztrófot közvetlen egymás mellett: `''`.

A bejárós ciklus

A lista a Python egyik olyan adattípusa, amelyben egyesével végig tudunk lépkedni az elemeken, azaz a lista bejárható, végigjárható. A bejárást egy ciklussal végezzük el, de nem a már megismert feltételes ciklussal.

Értelmezzük és futtassuk az alábbi programot!

A bejárós ciklus egyesével végiglépked egy bejárható objektum, például egy lista értékein. Az épp aktuális értéket betölti a ciklusváltozóba (esetünkben a „folyó” nevűbe). A ciklus magjában használhatjuk ezt a változót. A ciklusmag éppúgy, mint a while-ciklusnál, bentebb kezdődik.

Módosítsuk a kacatos programunkat úgy, hogy a végén bejárós ciklussal írjuk ki a kacatlistát!

Listák és bejárásuk

Feladatok

1. Írjunk olyan programot, amely egy verseny résztvevőinek célba érkezés szerinti névsorát kéri a felhasználótól, majd kiírja a dobogósokat és a sereghajtót!
   Minden versenyző bekérése előtt írja ki az épp aktuális névsort!

```
versenyzők = []

versenyző = None
while versenyző != '':
    print('A versenyzők jelenleg:', ', '.join(versenyzők))
    versenyző = input('Kérek egy versenyzőt! ')
    if versenyző != '':
        versenyzők.append(versenyző)

print('Az első helyezett: ', versenyzők[0])
print('Az második helyezett: ', versenyzők[1])
print('A harmadik helyezett: ', versenyzők[2])
print('A sereghajtó: ', versenyzők[-1])
```

2. Írj olyan programot, amely egy-egy listába bekéri három-három leves, főétel és desszert nevét, majd kiír három menüt, mindegyikben egy levessel, főétellel és desszerttel!

A `range()` függvény arra való, hogy számsorozatokat állítson elő. Háromféle módon használható:

- csak egy számot adunk meg a zárójelben: a `range(5)` a 0,1,2,3,4 számsort állítja elő,
- két számot adunk meg: a `range(2,5)` a 2,3,4 számsort állítja elő,
- három számot is megadunk: a `range(5,15,3)` az 5,8,11,14 számsort állítja elő.

A range-objektumokat leggyakrabban arra használják, hogy az előállított számsorozatot for-ciklussal bejárják. Minden, ami for-ciklussal megoldható, megoldható while-ciklussal is, de sok esetben elegánsabb és egyszerűbb így.

3. Írjuk ki az első 10 természetes számot és a négyzetüket!

```
for szám in range(10):
    print(szám, szám**2)
```

Írjuk át a programot úgy, hogy while-ciklust használjon!

4. Három egymásba ágyazott bejárós ciklussal rajzoljuk ki az alábbi ábrát!

```
ooooo
ooooo
ooooo
ooooo

ooooo
ooooo
ooooo
ooooo

ooooo
ooooo
ooooo
ooooo
```

```
1. for téglalap in range(3):
2.     for sor in range(4):
3.         for oszlop in range(5):
4.             print('o', end='')
5.         print('')
6.     print('')
```

Mi a szerepe az ötödik sornak, és mi a hatodiknak?
Hol kell átírni a kódot, hogy három, az alábbival egyező háromszöget rajzoljon?

```
o
oo
ooo
oooo
```

Ötlet: a `range()` függvény paraméterében szerepelhet ciklusváltozó is.

A lista adattípusnak van `remove()`, azaz eltávolít, kivesz tagfüggvénye is. Az `append()`-hez hasonlóan működik: a zárójelben kell megadnunk azt az elemet, amit kiveszünk a listából. Ha több azonos törlendő van, akkor a `remove()` tagfüggvény az elsőt fogja kivenni a listából. A `len()` (azaz hossz) függvény pedig arra használható, hogy egy lista elemszámát megadja.

5. Írjunk programot, amely egy autókölcsönző munkáját szimulálja! A kölcsönző a munkanapot egy listányi autóval kezdi, és addig kölcsönöz, amíg minden autót ki nem ad. A program írja ki az autók listáját, és kérdezze meg, melyiket kölcsönzi ki a felhasználó. Írja ki, hogy mik maradtak benn, és kérdezzen újra, és így tovább.

```
1. autók = ['Trabant', 'T-Modell', 'Rolls-Royce']
2.
3. while len(autók) > 0:
4.     print('Kölcsönözhető:', ', '.join(autók))
5.     mit = input('Melyik autót kölcsönzi ki? ')
6.     if mit in autók:
7.         autók.remove(mit)
8.     else:
9.         print('Ilyen autóval nem szolgálhatunk.')
```

Addig kölcsönzünk, amíg minden autó ki nem megy.

Így biztosítható, hogy ne akarjunk nem létező elemet kivenni a listából – abba belehalna a program.

Listák mindenféle adatokkal

Feladatok

1. Szimuláljunk tízmillió kockadobást, és az eredményeket tároljuk listában! A programunk számolja meg, hogy hányszor „dobtunk” hatost!

```
1.  import random
2.
3.  dobások = []
4.  for _ in range(10000000):
5.      dobás = random.randint(1,6)
6.      dobások.append(dobás)
7.
8.  ennyi_hatos = 0
9.  for dobás in dobások:
10.     if dobás == 6:
11.         ennyi_hatos += 1
12.
13. print('Összesen', ennyi_hatos' 'hatost dobtunk.')
```

Nem használjuk fel a ciklusváltozót, ezért jó ez a semmitmondó név.

Az első ciklus előállítja a dobásokat.

A második ciklus összeszámolja a hatosokat.

Számoljuk össze mind a hat lehetőség előfordulásait egy 1–6 között futó külső ciklussal!

2. Az egy elem előfordulásának megszámlálására a Python sokkal egyszerűbb megoldást kínál – a lista adattípus `count()` tagfüggvényét. Keressük meg az interneten, hogy miként kell használni, és próbáljuk ki!
3. Kihívást jelentő feladatok, ahol nem segít rajtunk a `count()`, és mindenképp be kell járnunk a listát:
   a. Hány helyen előzi meg a hatos dobást ötös dobás?
   b. Hány helyen van egymás után két hatos?

Eddig a listáinkat érték szerint jártuk be. Amikor listák index szerinti bejárásáról beszélünk, akkor a ciklusváltozóban nem az aktuális listaelem értéke van, hanem az aktuális listaelem sorszáma, azaz indexe.

Az index szerinti bejárásnak a Pythonban két módszere is van. Az első jobban közelít a – Pythonban nem létező, de más nyelvekben elterjedten használt – számlálós ciklusok használatához.

```
1.  fővárosok = ['Párizs', 'Bécs', 'Róma', 'Prága']
2.
3.  for index in range(len(fővárosok)):
4.      print(index, fővárosok[index])
```

A len() itt 4-et ad vissza, a range(4) pedig 0,1,2,3-at. Az index változó értéke tehát 0-1-2-3 lesz.

Kiírjuk a „fővárosok” lista nulladik, első, második és harmadik elemét.

A második módszer pythonosabb, és egyszerre kapjuk meg az aktuális elem indexét és értékét. Itt lényegében két ciklusváltozónk van.

```
1. fővárosok = ['Párizs', 'Bécs', 'Róma', 'Prága']
2.
3. for index, főváros in enumerate(fővárosok):
4.     print(index, főváros)
```

enumerate: számozd be!

Futtassuk a fenti kódot, és figyeljük meg a program kimenetét! Értelmezzük a negyedik sorban lévő két változó szerepét!

4. Állítsunk elő magunknak ezúttal egy tízelemű, pénzfeldobások eredményeit tartalmazó listát! Hány olyan eset van, amikor az aktuális és az előző dobás is „fej”?

```
1.  import random
2.
3.  feldobások = []
4.  for _ in range(10):
5.      feldobás = random.choice(['f', 'í'])
6.      feldobások.append(feldobás)
7.
8.  print('A feldobások:', ', '.join(feldobások))
9.
10. fej_után_fej = 0
11. for index, feldobás in enumerate(feldobások):
12.     if index > 0 and \
        feldobás == 'f' and feldobások[index-1] == 'f':
13.         fej_után_fej += 1
14.
15. print('Ennyiszer volt fej után fej: ', fej_után_fej)
```

Listában adjuk meg, hogy mik közül lehet választani.

Az első értéknél még nem tudjuk megnézni az előzőt.

A túl hosszú sorokat visszaperrel törhetjük. Te nyugodtan írhatod egybe a gépeden.

Az előző elem az, aminek eggyel kisebb az indexe.

5. Állítsunk elő harmincelemű, nulla és kilenc közötti véletlen számokat tartalmazó listát! A számok egy útvonal magassági adatait jelentik. Meredek az útszakasz, ha legalább kettővel magasabb az aktuális hely, mint az előző. Hány meredek szakasz van az úton? És visszafelé?
6. Kihívást jelentő feladat: A programunk elején adjunk meg két listát:
   - az első tartalmazzon öt filmcímet,
   - a második a filmek egy-egy főszereplőjét!

   Az első filmhez az első szereplő tartozik, a másodikhoz a második, és így tovább.
   Írjuk ki a filmcímeket, majd az egyik, véletlenszerűen kiválasztott szereplőt! Kérdezzük meg a felhasználótól, hogy a kiírt szereplő melyik filmnek a főszereplője! Értékeljük a válaszát!

7. Kihívást jelentő feladat: Állítsunk elő nyolcvanelemű, -5 és 3 közötti egész számokból álló listát! A számok egy úszó palackorrú delfin magasságát jelentik. A delfin ki-kiugrál a vízből, ilyenkor pozitív a magassága. Nulla a magasság, amikor a felszínen úszik, negatív, amikor a víz alatt. Írjunk programot, ami választ ad a következő kérdésekre!
   a. Az út mekkora részét tette meg a delfin a vízben, illetve a víz alatt? A válaszok megadhatóak törtszámként és százalékként is.
   b. Víz alatt, vagy víz felett volt-e többet a delfin? A vízfelszínen való utazás egyik esetbe sem számít bele.
   c. Milyen hosszú volt a leghosszabb kiugrása? Az út hányadik pontjánál kezdődött?
   d. Hányszor törte át a vízfelszínt, azaz hányszor követ a listában negatív számot pozitív, vagy fordítva?
   e. Mély merülésnek számít, ha a delfin -4-es vagy -5-ös mélységben van. Az út során hányszor merült mélyre? Figyeljünk arra, hogy például a 4 -2, -4, -5, -5, 3 útvonal csak egy mélyre merülést jelent!