

# P Algoritmizálás és programozási nyelv használata

## Mi az a programozás?

### Mi a program?

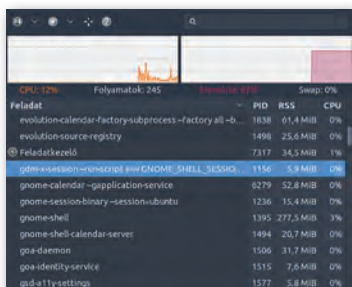
A számítógép, a telefon, az összes olyan eszközünk, amiben valamilyen számítógép van, önmagában képtelen ellátni azt a feladatot, amire készült. Csak egy darab „vas” – azaz hardver. Csak akkor képes igazán működni, ha fut rajta egy (vagy sok) program, alkalmazás – azaz szoftver. Szoftver, program, alkalmazás – 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.

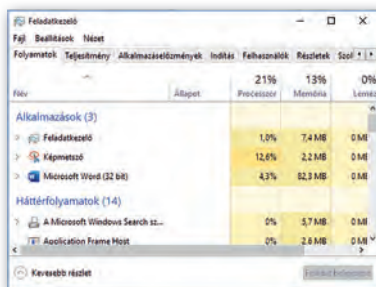
Az informatikaórán bennünket jobban érdekelnek a hagyományos értelemben vett számítógépek (laptopok, asztali gépek, szerverek) és a mobil eszközök. Ezek csak bekapcsoláskor futtatnak egyetlen programot, ami azt mondja el nekik, hogy honnan és hogyan kell betölteniük a „fő” programjukat: az operációs 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 egerrel 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.



Feladat	PID	RSS	CPU
evolution-calendar-factorysubprocess--factory-all--b...	1838	61.4 MiB	0%
evolution-source-registry	1498	25.6 MiB	0%
Feladatkezelő	7317	34.5 MiB	1%
gnome-karaoke-szoftver-entertainment-shell-session	1156	5.9 MiB	0%
gnome-calendar--application-service	6279	52.8 MiB	0%
gnome-session-binary--session--ubuntu	1236	15.4 MiB	0%
gnome-shell	1395	277.5 MiB	3%
gnome-shell-calendar-server	1494	20.7 MiB	0%
gpa-danmon	1506	31.7 MiB	0%
gpa-identity-service	1515	7.6 MiB	0%
gpa-identity-service	1577	5.8 MiB	0%



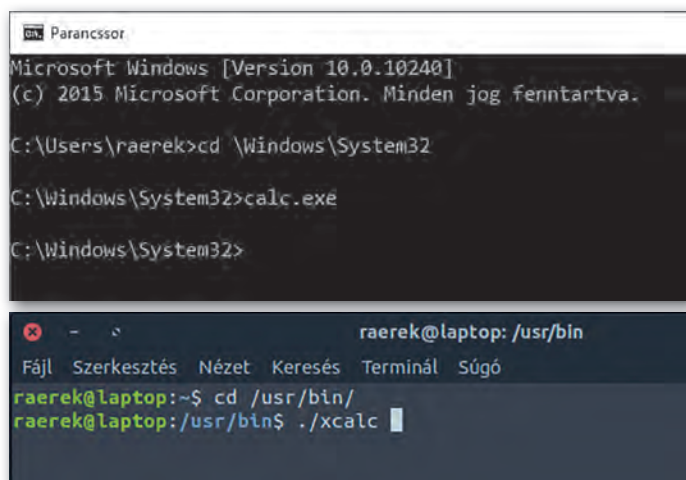
Feladat	CPU	Memória	Privát
Alkalmazások (3)			
Feladatkezelő	1.0%	7.4 MB	0 MB
Fájlmegosztás	12.6%	2.2 MB	0 MB
Microsoft Word (32 bit)	4.3%	82.3 MB	0 MB
Háttérprogramok (14)			
A Microsoft Windows Search is...	0%	5.7 MB	0 MB
Application Frame Host	0%	2.6 MB	0 MB

- ▶ 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!



```
Parancssor
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. Minden jog fenntartva.

C:\Users\raerek>cd \Windows\System32

C:\Windows\System32>calc.exe

C:\Windows\System32>

raerek@laptop: /usr/bin
Fájl Szerkesztés Nézet Keresés Terminál Súgó
raerek@laptop:~$ cd /usr/bin/
raerek@laptop:/usr/bin$ ./xcalc
```

► 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 krikzkrakszot 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 teen-dőit. 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!



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. Gépígye 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 legelső sora kötelezően

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

legyen. Erre Windows alatt nincs szükség.

Mentsük el a fájlnkat – érdemes a programjainknak létrehozni egy mappát –, és figyeljünk rá, hogy a fájl kiterjesztése `.py` legyen, azaz a fájl teljes neve legyen például `első.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` parancssal lépünk abba a mappába, ahova a fájlt mentettük. Linuxon és macOS-en még futtathatóvá kell tennünk a fájlt a `chmod +x első.py` parancs kiadásával. Ezt követően futtathatjuk programunkat a program nevét beírva.

```

C:\Users\raerek>cd programjaim
C:\Users\raerek\programjaim>első.py
Szia.

raerek@asuslaptop: ~/programjaim
Fájl Szerkesztés Nézet Keresés Terminál Súgó
raerek@asuslaptop:~$ cd programjaim/
raerek@asuslaptop:~/programjaim$ chmod +x első.py
raerek@asuslaptop:~/programjaim$ ./első.py
Szia
raerek@asuslaptop:~/programjaim$

```

► 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 kóduvá 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 téltlent jelent, holott mi épp itt fogunk sokat ténykedni.

Amikor az IDLE elindult, egy úgynevezett Python Shellt látunk, amiben szintén lefutathatók 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 felejtünk 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 szöveggként 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 szöveggként 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 `Ma_jom`, `ma_jom` és `ma_joM` 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 tudja 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 futtasuk le! Azt látjuk, hogy a program vár. Ha nyomkodjuk a billentyűket, akkor amit lenyomtunk, kiíródik, ha `Enter`t 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)
```

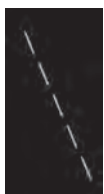
```
1. autó = 'Trabant'
2. autó = 'Renault'
3. autó = 3 - 5
4. print(autó)
```

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 írunk – a `print('\\\\')` parancs csak egy visszapert ír ki.

3. Rajzoljuk ki az alábbi ábrákat karakterek használatával!



4. Kérdezzük meg a felhasználótól (a program használatjától) a vezetéknévét! Kérdezzük meg a keresztnévét 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 figyelniünk a szóközökre, ezért alakul át a negyedik sor többi része is.

```
1. vezetéknév = input('Mi az Ön becses vezetéknéve?')
2. keresztnév = input('Érdeklődhetek a keresztnéve felől is?')
3. print('Üdvözlöm,', vezetéknév, keresztnév, '!')
4. print('Üdvözlöm,', vezetéknév, ' ', keresztnév, '!', sep='')
```

Figyeljünk a vesszőkre!

Ezek új szóközök, az  
aposztrófokon belül.

## 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.')
```

*beszédés változónév, amiben  
nincs szóköz*

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 születtedél: ', 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 + cercófok

Remélhetőleg nagyjából ezeket mondtuk:

- Adjuk össze a két számot, és az eredményt tegyük a „szám” változóba!
- Olvassuk ki az orangutánok és a cercófok 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)` sortehát 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: ''
```

Nem adunk meg értéket, csak Entert nyomunk.

```
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'
```

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 „összeadód-nak” 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? ')
```

Itt ismét  
karakterlánc a  
felhasználó kora.

Érdeemes lehet  
megszoknunk a  
„kozmetikai” szóköz  
használatát.

## 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 távolságadat 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 vezeté- 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 szöveget 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 írunk 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 száma

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 **flowchart**ra 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.

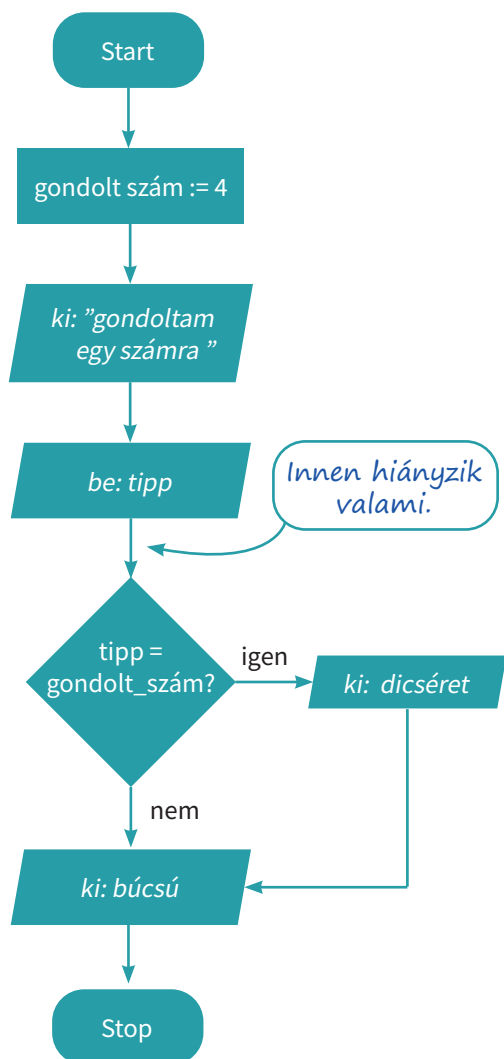
1. Mit csinál a program?
2. Mi az a lépés, amit nem tüntettünk fel? (Ha most nem jövünk rá, nem probléma: hamarosan úgyis megírjuk a programkódot, akkor szólni fog a Python.)
3. Hogyan olvassuk ki a `:=` jelet? Mi a megfelelője Pythonban?

Miközben a programunkat flowchart-rával megfogalmazzuk, **algoritmust** (receptet) adunk a bennünket érdeklő probléma megoldására. A flowchartok elég látványosak, de hamar lelopnának a könyvlapról, így aztán a gyakorlatban gyakrabban használunk egy másik algoritmusleíró eszközt, a **mondat szerű leírást**.

A leírás szabályaira rá fogunk érezni.

4. Vessük össze ezt a leírást a flowchart-jával!
5. Melyik az a művelet, ami a mondat szerű leírásban már megvan, de a flowchart-jában még hiányzik?

```
gondolt_szám := 4
ki: „gondoltam egy számra”
be: tipp
tipp átalakítása egészszé
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. Típpeld 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 helytelenlennel 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)!

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!

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

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

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

Ez itt az elágazás FELTÉTELE.

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. Észrevevessük, hogy kétféleképp lehet egyet tévedni: vagy eggyel nagyobbat, vagy eggyel kisebbet tippelve. Az „eggyel nagyobbat tippelt” így írható le:

```
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
```

```
tipp = gondolt_szám+1.
```

A másik feltétel megfogalmazása nem okozhat gondot, de ezek szerint két feltétellel 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. Lévé 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_száma = 4
2. tipp = input('Gondoltam egy számra. Típpeld meg! ')
3. tipp = int(tipp)
4. if tipp == gondolt_száma:
5.     print('Ügyes!')
6.     print('Gratulálok.')
7. elif tipp == gondolt_száma + 1 or tipp == gondolt_száma - 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_száma + 1 or gondolt_száma - 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 Python-ban 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 tölteni a programmal a `random` nevű modult. A programunk első sorai a következőképp alakulnak:

```

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

```

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

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

1 és 6 közötti egész számot állítunk elő.

## Elágazások és véletlenek

### 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 nagyobbbat!
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_szöveggént = str(egyik)
6. másik_szöveggént = str(másik)
7. tipp = input('Mennyi ' + egyik_szöveggént + ' 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 vastaggal szedve):

```
Mi az egyik csapat neve? Tóparti királyok
Hány pontot szerzett? 78
Mi a másik csapat neve? Talpasi csodatévők
Hány pontot szerzett? 54
Az összeadás 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”

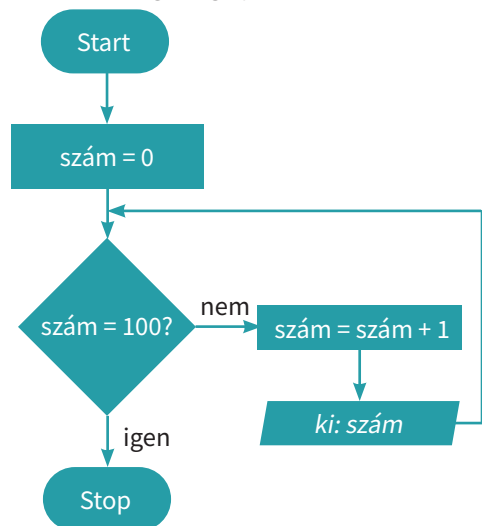
6. Ha bonyolultabb összetett feltételeket fogalmazunk meg, érdemes lehet zárójelek használatával egyértelműsíteni a szándékunkat. Melyik feltételmegfogalmazás helyes az alábbiak közül?
  - a. Ha (van tollunk és van ceruzánk) vagy van egy papírlapunk: tudunk rajzolni valamit.
  - b. Ha (van tollunk vagy 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 tortaeszélék,
  - 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 „binnen 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.')
```

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

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

## Túltenni Gauss

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, még hozzá 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_száma = random.randint(1,6)
4. kitalálta = False
5. while not kitalálta:
6.     tipp = int(input('Szerinted? '))
7.     if tipp == gondolt_száma:
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 figyelniük 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_száma = 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_száma:
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. Diophantos ókori görög matematikus verses sírfelirata több fordításban fellelhető az interneten. A felirat alapján fogalmazz meg egyenletet, és írd programot, ami megoldja! Hány évesen halt meg Diophantos?
  - 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 visszapér és egy `n` betű. A visszapér (backslash) szól, hogy a következő betű nem is betű, hanem vezérlő-karakter, az `n` pedig azt jelenti: new line, új sor. Ezt azonban felülbíráthatjuk.

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

A két ciklusnak két külön számlálója van.

Minden sor elején újakezdjük a csillagok számolását.

Ez a három programsor ír ki egy sornyi csillagot.

A belső ciklus magja nagyobb behúzást kap.

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

- 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

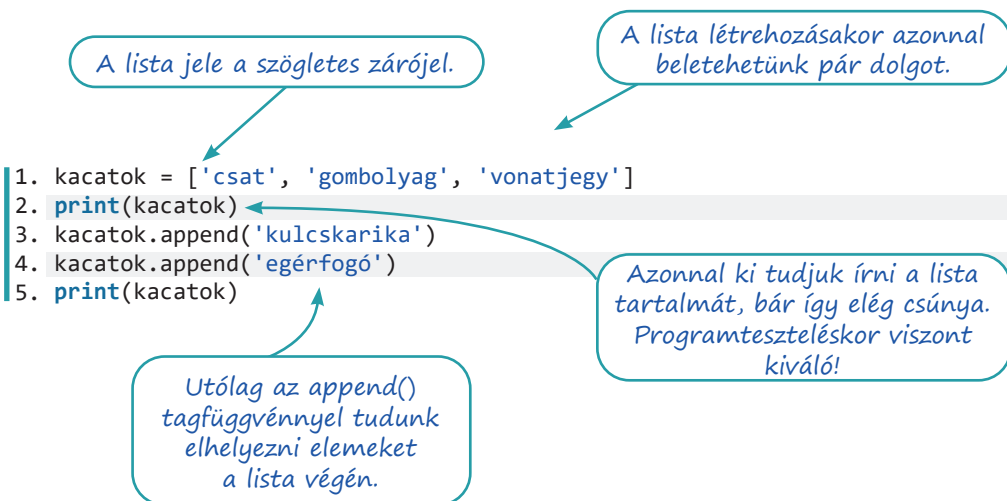
A programjainkban az adatokat változóban tároljuk, és eddig négy változótípust, adattípust ismerünk:

- 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 (szöveggént), évével (egéssként), hónapnevével (szöveggént) és napjával (egéssként), valamint a győztes oldallal (szöveggént).

## A listaelemek sorszámozása

Az adatok sorszámozást 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 sorszáma. Így az előző lista nulladik eleme 'Isaszeg', a harmadik pedig 6. A sorszámsra 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 megnyugtadjuk, hogy lehet egy lista eleme egy másiknak. Aki ettől megijed, azt is megnyugtadjuk: 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 elsősorban talán meglepő. A zárójelben átadott lista elemeit fűzi össze egyetlen karakterláncná, az elemek közé pedig az elején, aposztrófok között megadott karakterláncot teszi.

Egészítsük ki az előző programunkat az alábbi sorokkal!

A kacsatok közé vessző  
és szóköz kerül.

```
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!



A listáknak érdemes többes számú főnevet névül választani.

Üres listát hozunk létre: csak két szögletes zárójel.

Kezdeti értéket kell adnunk, hogy egy sorral lentebb megvizsgálhassuk az értékét.

```
1. kacatok = []
2.
3. kacat = 'bármí'
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='')

```

Csak akkor illesztjük a lista végére, ha nem „elfogyott”.

Ha utoljára azt mondta a felhasználó, hogy „elfogyott”, nem kérdezzünk többet.

A harmadik sorban csak azért adjuk a „bármí”-t a kacat változó értékéül, hogy egy sorral lentebb megnézhessük, hogy az érték nem „elfogyott”-e. A „bármí” helyett akármi más is szerepelhet itt, csak az „elfogyott” nem, mert akkor egyszer sem lépnék be a ciklusba. A „bármí” 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.

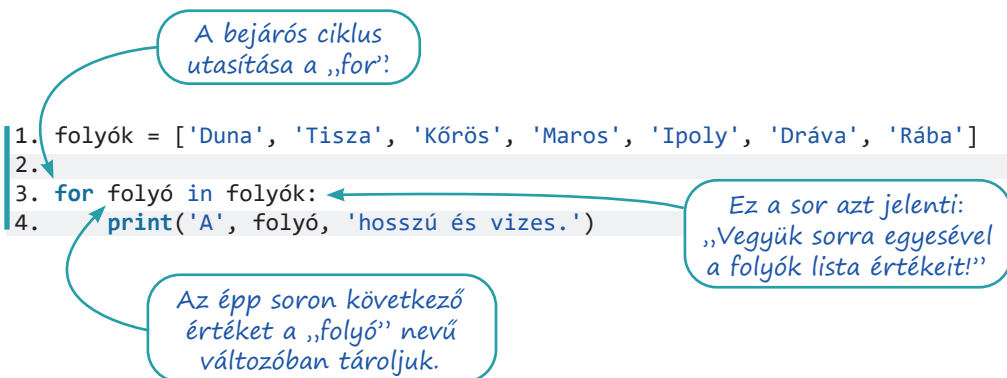
Informatikusszemmel nézve nem túl szerencsés, amit a hatodik (és később a negyedik) sorban művelünk. Itt ugyanis onnan tudjuk, hogy nincs több kacat, hogy egy különleges nevű kacatot adunk meg. Ha egyszer véletlenül a felhasználónak mégis lesz „elfogyott” nevű kacatja, nem tudja bevinni a programunkba.

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!

```
1. versenyzők = []
2.
3. versenyző = None
4. while versenyző != '':
5.     print('A versenyzők jelenleg:', ', '.join(versenyzők))
6.     versenyző = input('Kérek egy versenyzőt! ')
7.     if versenyző != '':
8.         versenyzők.append(versenyző)
9.
10. print('Az első helyezett: ', versenyzők[0])
11. print('Az második helyezett: ', versenyzők[1])
12. print('A harmadik helyezett: ', versenyzők[2])
13. 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 számsorozatot 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!

```
1. for szám in range(10):
2.     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 járunkunk 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', 'i'])
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 \
13.         feldobás == 'f' and feldobások[index-1] == 'f':
14.         fej_után_fej += 1
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 filmcímét,
  - 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 választát!

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 esetben 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 hány-szor merült mélyre? Figyeljünk arra, hogy például a 4 -2, -4, -5, -5, 3 útvonal csak egy mélyre merülést jelent!



## Mobil informatikai eszközök

A technika gyors fejlődése a mindennapi életünkben is sok változást hoz, így van ez az informatika világában is. Az informatikai eszközök köre jelentősen kibővült. Kialakult az informatika egy új ága, amelyet mobil informatikának nevezünk.

A mobil informatikai eszközök közé sorolhatunk minden olyan eszközt, amely a számítógéphez hasonlóan működik, de hordozható. Az ilyen eszközök rendelkeznek a számítógép alapvető hardverelemeivel. Tartalmaznak processzort, memóriát, kimeneti és bemeneti eszközöket. Működésüket általában operációs rendszer biztosítja. Hordozhatóságuk miatt azonban az ilyen eszközök nem vagy nemcsak hálózati áramforrásról működnek, hanem akkumulátorral rendelkeznek. A számítógépes hálózathoz való csatlakozásuk elsősorban vezeték nélkül, wifikapcsolaton vagy mobilhálózaton keresztül történik.

A mobil eszközök közé sorolhatók a tabletek, laptopok, okostelefonok, e-book-olvasók, okosórák. Körük az újabb technológiák és megoldások fejlődésével rohamosan bővül. Használatuk, új technikai megoldásaik a mindennapi életünkre is hatással vannak.

A hagyományos asztali számítógépektől működésükben a laptopok különböznek a legkevésbé. Operációs rendszereik, szoftvereik az asztali számítógépekétől nem különböznek. Hardverük felépítése is csak annyiban, ami a hordozhatóság érdekében fontos, például kisebb méretű, alacsonyabb energiaigényű alkatrészekre van szükségük.

A tablet vagy táblagép nagyobb méretű érintőképernyővel rendelkező eszköz, amelynek nincs billentyűzete. Dokumentumok szerkesztésére, hosszabb munkára kevésbé alkalmas, de nagy képernyője miatt különböző médiatartalmak kényelmesen megjeleníthetők rajta. Operációs rendszerük Android, iOS és Windows is lehet. Teljesítmény és felszereltség szempontjából az ilyen gépek széles skálán mozognak. Speciális toll segítségével kézírással is írhatunk az érintőképernyőre. Könnyű és keskeny, ezért jól hordozható eszköz.

Az e-book-olvasók a hagyományos könyvek alternatívái. A legtöbb papíralapú könyvnél lényegesen vékonyabbak és kisebbek. Elektronikus formában tárolják a könyveket, ezért az ilyen eszközökön akár több könyvespolcnyi



► Tablet, laptop, okostelefon



► Nyomtatott könyv és e-book olvasó

könyvet is magunkkal vihetünk. A tabletekkel ellentétben az e-book-olvasók többnyire e-tinta- (e-ink) technológiát használnak a kép megjelenítésére. Ennek jellemzője, hogy nincs háttérvilágítás, ezért olvasásuk az emberi szem számára sokkal kíméletesebb, mint a számítógépeké vagy tableteké, és az akkumulátor készenléti ideje is jelentősen hosszabb.

Az **okosórák** legtöbbször a használói mobiltelefonjaihoz kapcsolódnak. Az idő jelzésén kívül számos funkcióval rendelkeznek. Általában alkalmasak a tulajdonos egészségügyi adatainak, sporttevékenységének nyomon követésére, esetleg zenelejátszásra, elektronikus fizetésre. Jelzik a telefonra beérkező hívásokat, üzeneteket, ezeket bizonyos modellek-nél el is lehet indítani róluk.

## Az okostelefonok

A legszélesebb körben használt mobil informatikai eszköz a **mobiltelefon**. A mobiltelefont eredeti funkciója szerint csak telefonálásra és üzenetküldésre használhattuk. Idővel a fejlesztések révén kiegészült az internetes kommunikáció lehetőségével. A ma használt mobiltelefonok többsége **okostelefon**, amelyen operációs rendszer működik, és az alapfunkciók mellett egyéb programok futtatására is alkalmas. Jellemzőjük, hogy érintőképernyővel, kamerával rendelkeznek, és a szöveges adatokat legtöbbször virtuális képernyő-bilentyűzeten vihetjük be.



► Az okostelefon sok feladatra alkalmas

Az okostelefonokat a hardverük, az operációs rendszereik és a rajtuk futó alkalmazások fejlődése révén az életünk egyre több területén használjuk. A telefonunk ma már nagyon sok funkcióval rendelkezik. Alkalmas internetböngészésre, fénykép- és videókészítésre, zenehallgatásra és filmnézésre. Ezért sok régebben külön-külön megjelenő eszközt helyettesítünk vele. Szinte bármilyen online kommunikáció folytatására használhatjuk, kezelhetjük az e-maileket, az internetalapú beszélgetéseket. Meg tudunk velük nyitni dokumentumokat, sőt sok esetben

szerkeszthetjük is azokat. Személyes adataink nagyon nagy részét tároljuk ezeken az eszközökön. A rájuk telepített alkalmazások segítségével alkalmasak a mindennapi ügyeink intézésére, például banki vagy közüzemi szolgáltatást vehetünk igénybe. A telefonok jelentős része alkalmas arra, hogy bank- és egyéb kártyáinkat, utazáshoz kapcsolódó jegyeinket rajtuk tárolhassuk. Sok modellel a bankkártya használatát kiváltva tudunk fizetni.

Sokunk számára az okostelefon ma lényegesen több, mint egy egyszerű kommunikációs vagy informatikai eszköz. Tekinthejük a személyi asszisztensünknek, a munkaeszközünknek, az egyik legszemélyesebb tárgyunknak. Ezért nagyon fontos, hogy használatakor a biztonságra oda kell figyelnünk.

Az okostelefonok hardvere különbözik a számítógépektől, és az operációs rendszereik is eltérőek. A mobiltelefonokon két elterjedt operációs rendszer működik, az Android és az iOS rendszer. Van még néhány olyan operációs rendszer, amellyel ritkábban találkozunk a telefonokon. Ilyen a Windows Phone, Symbian és BlueberryOS.

Az iOS operációs rendszer az Apple cég iPhone telefonjaiban és iPad tableteiben fut. Az operációs rendszer kifejezetten ezekre a készülékekre készül, ezért jól meghatározhatók azok a hardverváltozatok, amelyeken működniük kell. Az iOS zárt forráskódú operációs rendszer, csak az Apple cég fejleszti. Az operációs rendszert használó telefonok túlnyomó többsége rendszeresen megkapja a szoftver frissítésének lehetőségét.

Az Android operációs rendszert a Google cég fejleszti. Az Android rendszert futtató telefonok, tabletek nagyon sokfélék. Sok gyártó eltérő felépítésű, felszereltségű hardverén kell működni a rendszernek. Ez az egyik oka annak, hogy az operációs rendszer nyílt forráskódú. Minden gyártó valamilyen a saját gyártmányaihoz alakíthatja, így biztosítva a jobb teljesítményt. Az Androidot futtató telefonok kisebb része kap rendszeres szoftverfrissítést. A telefon gyártója kezében van az adott modellek szoftverfrissítése, amely nem feltétlenül akkor történik, amikor a Google az új verziót közzéteszi, és nem minden modellre érhető el. Míg az iOS-nél az aránylag régebbi eszközökön is frissíthető az operációs rendszer, addig az Androidnál ez gyakran hiányzik.

A telefonok operációs rendszereire rengeteg applikációt telepíthetünk. Az applikációk egy része ingyenes, másokért fizetni kell. Az ingyenes alkalmazások nagy részében reklámok jelennek meg, illetve alkalmazáson belüli vásárlási lehetőséget ajánlanak fel, ezzel teszik gazdaságossá készítésüket. Az applikációk, függetlenül az áruktól, a megfelelő online áruházban kereshetők, vásárolhatók meg és tölthetők le. Az Android rendszer esetén ez a Play Áruház, az iOS esetén az App Store, a Windowsnál pedig a Microsoft Store.

A telefonos operációs rendszerekre jellemző, hogy az applikációkat ikonokkal jelenítik meg a képernyőn, és gesztusokkal irányíthatjuk ezeket. Gesztusoknak a képernyő érintésekor végzett különböző műveleteket nevezzük. Ilyen például a koppintás, a dupla koppintás, a legyintés, a húzás és a két ujjal való méretezés.



► Mobil operációs rendszerek logói

## Kérdések, feladatok

1. Gyűjtsünk össze olyan informatikai vagy hétköznapi feladatokat, amelyeket gyakran végzünk mobil eszközökkel!
2. Hasonlítsuk össze a hagyományos könyvet az e-könyvvel! Milyen előnyöket találhatunk az egyik vagy a másik használatában? Miben különbözhet az e-könyv e-book-olvasóval vagy tablettel történő olvasása?
3. Milyen feladatokat láthat el az okosóra? Miért lehet hasznos fiataloknak, és miért idősebbeknek?

## Az okostelefonok biztonságos használata

A telefonunkban egyre több személyes adatot tárolunk. Ezek között vannak olyan adatok, amelyeket mi magunk mentünk el, ilyenek például a névjegyeink, a jelszavaink, ezen keresztül az applikációkban tárolt adataink. Vannak olyan adataink is, amelyeket a telefonra telepített alkalmazások a beépített szenzorok segítségével gyűjtenek rólunk, anélkül hogy ennek tudatában lennénk. Ilyenek például a helyadatok, amelyek alapján követhető szinte minden lépésünk. Ezeket az adatokat használják az útvonaltervező programok a forgalmi dugók figyelésére. A személyes adatok között sok olyan akad, ami nem jó, ha illetéktelen kezekbe kerül.

Mi az, amire érdemes ügyelnünk a biztonság érdekében?

Az okostelefonok lényegében számítógépek, ezért rájuk is kerülhetnek **vírusok** vagy más **kártékony szoftverek**. Az ilyen kódokat leggyakrabban új applikációk telepítésekor szerezhetjük be. Az applikációk tartalmazhatnak olyan kódokat, amelyek az eszközön folyó tevékenységet figyelik meg, és az adatokat továbbküldik idegen félnek. Ezt végezhetik úgy, hogy a felhasználó tudomására hozzák, hogy az alkalmazás javítása érdekében teszik ezt. Az ilyen tevékenységet általában meg lehet tiltani. Vannak viszont olyan alkalmazások, amelyek ezt titokban végzik, nem tudjuk, milyen adatainkat gyűjtik, és mire használják fel az információkat.



► Play Áruház és App Store

A hivatalos áruházakba az applikációk csak ellenőrzés után kerülhetnek be. Ezért ha ezekből töltünk le, az jóval biztonságosabb. Az App Store szabályzata szigorúbb, mint a Play Áruházé, ezért az Android-felhasználóknak érdemes óvatosabbnak lenniük. Nem ajánlott a hivatalos áruházi verzió helyett más forrásból, esetleg ingyenesen beszerezhető applikációkat telepíteni. Ezek sokszor pont azért ingyenesek, mert a kívánt hasznot tudtuk nélkül, az adataink megfigyelésével, továbbadásával szerzik meg. A rendszer figyelmeztetéseit érdemes komolyan venni. Az áruházakban az

alkalmazás telepítése előtt tudunk tájékozódni. Ajánlott elolvasni az alkalmazás részletes leírását, figyelembe venni az értékelését, és az értékelők által írt kommenteket.

Sok alkalmazás már a telepítésekor vagy az első indításakor **engedélyeket** kér az eszközünk különböző adatainak, például tárolózkodási hely, kamera, névjegyek, mobil adatforgalom használatához. Az engedélykérést érdemes elolvasni, és átgondolni, valóban szükség van-e mindenre. Arra érdemes engedélyt adnunk, ami valóban hasznos funkciót biztosít számunkra. Az ilyen engedélyek a biztonsági kockázat mellett nem kívánt adatforgalmat is generálhatnak, ezzel anyagi kárt okozhatnak. Ajánlatos a beállításokban az alkalmazások engedélyeit időről időre felülvizsgálni.

A mobil eszközök sérülékenysége a hordozhatóságuknak is köszönhető. Egy telefont, tabletet sokkal könnyebb elveszíteni vagy eltulajdonítani, mint egy asztali számítógépet. Erre a kockázatra fel kell készülni. Az adatok védelme, az illetéktelen hozzáférés megakadályozása érdekében, mindig legyen az eszközön **képernyőzár**. Ez az előre beállított idő eltelte után csak valamilyen biztonsági kód megadásával engedi az eszközhöz való hozzáférést.

A képernyőzár feloldásához a készülékek különböző lehetőségeket biztosítanak. Ilyenek lehetnek: a PIN-kód (Personal Identification Number = személyi azonosító szám) vagy a képernyőre rajzolható minta. Az újabb eszközök lehetővé teszik a **biometrikus azonosítást**. Az erre alkalmas eszközök ujjlenyomatolvasóval, arcfelismerővel rendelkeznek. Ezek az egyedi azonosítók biztonságosabbak, mint a kódok, amelyek könnyen kifigyelhetők, esetleg visszafejthetők.



► Azonosítás ujjlenyomattal

Az eszközünk elvesztése a rajta tárolt adataink elvesztésével járhat. Fontos, hogy legyen ezekről **biztonsági másolatunk**. Ezt készíthetjük saját magunk is, de a mobil operációs rendszerek mindegyike tartalmaz ehhez kapcsolódó szolgáltatást. Az adatainkat általában a rendszerhez kapcsolódó személyes felhőbe mentik (Android – Google Drive, iOS – iCloud, Windows – OneDrive). Ezekből eltulajdonítás, de egy esetleges készülékváltás esetén is könnyen helyreállíthatók a készülékhez kapcsolt adataink.

A készülék eltulajdonítása esetén jelent segítséget a **nyomkövetés** beállítása. Ennek bekapcsolásával lehetőség van az eszköz megkeresésére, távolról történő zárolására és a rajta tárolt adatok törlésére.

Az egyes alkalmazások használatakor igyekezzünk figyelni a belépés biztonságára. Gondoljuk át, melyek azok az alkalmazások, amelyekbe belépve szabad maradnunk. Indokolt esetben érdemes minden alkalommal az újra belépést választani. Ennek megkönnyítésére sok eszköz állhat rendelkezésünkre. Több alkalmazásban beállítható a biometrikus azonosítás. Vannak olyan alkalmazások, amelyekkel a jelszavainkat tudjuk tárolni. A **jelszókezelők** titkosítva tárolják a bejelentkezési adatokat, jelszavakat, esetleg más fontos adatokat (pl. bankszámlaszámokat). Ilyen jelszókezelő például a LastPass vagy az iCloud Kulcskarika.

Érdemes beszélnünk az okostelefon-használat más irányú veszélyeiről is. Napjainkban az emberek, és főképpen a fiatalok egyre több időt töltenek mobiltelefon, tablet használatával. Bizonyos határt átlépve ennek lehetnek a testi és lelki egészségre nézve káros hatásai. Figyeljünk a mobil eszközök mellett töltött időre! Ezt az időt az operációs rendszer vagy egyéb applikációk segítségével nyomon követhetjük. Érdemes az eszközt bizonyos időközönként letenni. A telefonos értesítések állandó hangjelzései szintén lehetnek zavaró hatásúak. Ajánlott az értesítéseket, hangjelzéseket legalább az éjszakai pihenés idejére elnémitani.



► A túlzott mértékű telefonhasználat káros lehet

## Kérdések, feladatok

1. Milyen feltételek mellett használhatjuk a biometrikus azonosítást?
2. Milyen káros hatásokkal járhat a túlzott mobil eszköz-használat? Hogyan lehet ezt megelőzni?
3. Tekintsük át a mobiltelefonunk beállításait! Mely applikációk milyen jogosultsággal rendelkeznek? Milyen biztonsági beállításokat, appokat találhatunk a mobiltelefonunkon? Hasonlítsuk össze a különböző operációs rendszerek lehetőségeit!



## Mobiltanulás

A mobil eszközöket életünk egyre több területén használjuk. A kapcsolattartás, kommunikáció, szórakozás, különböző ügyintézés hatékonyágnövelése vagy az utazásszervezés mellett több olyan szolgáltatást adnak a mobil eszközök, amelyek a tanulásunkat, ismeretszerzésünket segíthetik, hatékonyabbá tehetik. A mobil eszközök funkcióit hasznosító tanulási formákat nevezzük mobiltanulásnak (M-learning). A mobil eszközeink segítségével új ismereteket szerezhethetünk, gyakorolhatunk, elmélyíthetjük a tudásunkat, szemléletesebbé, érthetőbbé tehetjük a tananyagot, vagy segíthetjük egymást a tanulási folyamatban.

A mobiltanulás eszközeinek köre igen széles, és állandóan változik, bővül. Alapos áttekintésük meghaladja a könyv kereteit, itt csak néhány példát mutatunk be.



Mobiltelefont tanórán mindig csak tanári engedéllyel, az iskolai házirend betartásával szabad használni.

### Általános tanulást segítő alkalmazások

Az alkalmazások egy része általánosan a tanulási folyamat segítésére használható, míg mások konkrét tantárgy tanulásához alkalmazhatók.

### Jegyzetelés, megosztás

Sok olyan alkalmazás áll a rendelkezésünkre, amelyek telefonon is használhatók, és segítenek az online együttműködésben, információörögzítésben és -megosztásban. Ezek az alkalmazások telefonon, tableten is futnak, de asztali gépen is. Billentyűzettel vagy kézírással készíthetünk jegyzeteket. A jegyzet nemcsak írásos anyagot, hanem képeket, rajzokat, videókat, linkeket is tartalmazhat. Egyes tabletek képernyőjére speciális tollal írhatunk, így az jóval pontosabb, a kézíráshoz hasonló kezelést tesz lehetővé. Mobil eszközökön használható jegyzetelő alkalmazás például a OneNote, az Apple jegyzetek, az Evernote és az Inkscape. Bizonyos applikációk megosztható táblaként működnek. Ilyen például a Whiteboard alkalmazás. Szerkesztés közben többen írhatunk rá, a dokumentumot közösen, együttműködve alakíthatjuk ki. Ha megfelelő felszereléssel rendelkezünk, akkor ezekkel az alkalmazásokkal a hagyományos papíralapú jegyzetelés teljesen kiváltható. Sok alkalmazás a kézírást is képes felismerni és átalakítani nyomtatott szöveggé. A digitális jegyzeteknek számos előnyük lehet. Kisebb helyen tárolhatók, nagy mennyiségben is magunkkal vihetjük őket. A rendszerezésük, a tartalmukban való keresés egyszerűbbé válhat.



► Digitális jegyzet készítése

Gondolattérkép-készítők (pl. XMind), projektszervezők (pl. Trello), alkalmazás- és linkgyűjtemények (pl. PearlTrees) szintén segíthetnek bennünket a tanulás során.

## Kép-, filmkészítés, szerkesztés

A mobil eszközök sokszor kamerát tartalmaznak, így szoftvereikkel fotó, videó- és hangfelvétel készíthető. A tanulási folyamat során a felvétel sok esetben segítséget jelenthet. Felvehetünk velük egy-egy előadást, eljárást, mozdulatsort azért, hogy újra megnézhessük, lejátszhassuk, könnyebben memorizálhassuk.

Alkalmas lehet egy-egy fizikai, kémiai kísérlet felvételére. A videó segítségével pontosabban megfigyelhetjük, elemezhetjük a jelenséget, mint szabad szemmel történő megfigyeléskor. A videót egy gyorsan lejátszódó folyamat esetén lassíthatjuk, egy lassú folyamat esetén gyorsíthatjuk, vagy megállíthatjuk egy-egy fontos részletnél. A videót átalakíthatjuk, vághatjuk, feliratozhatjuk, szerkeszthetjük.

A videók készítéséről és szerkesztéséről részletesebben a *Multimédiás dokumentumok készítése* című fejezetben olvashatunk.

## Oktatóprogramok

Sok kifejezetten oktatás céljára készített program áll rendelkezésünkre, melyek jelentős része mobil eszközökön is, vagy csak ott használható. Az online kvízek (Kahoot, Quizlet, Quizizz...), tesztek (Redmenta, Socrative, Forms, Google űrlap...), feladatmegoldó szoftverek (LearningApps, OkosDoboz...) szinte mindegyikéhez használhatunk mobil eszközt. Így olyan helyzetben is dolgozhatunk velük, ahol nem áll rendelkezésre asztali számítógép.

### Kiterjesztett valóság (Augmented Reality, rövidítve AR)

Az oktatást segítő alkalmazások egy része **kiterjesztett valóság** segítségével teszi szemléletesebbé a tananyagot.

A kiterjesztett valóság a körülöttünk lévő valós teret virtuális elemekkel egészíti ki. Például mobil eszköz kameráján keresztül nézve olyan tárgyakat is láthatunk a környezetünkben, amelyek valójában nincsenek ott. Ellentétben a virtuális valósággal, a kiterjesztett valóság alkalmazásainak nem feltétlenül van szükségük különleges eszközre, legtöbbjük kamerával, GPS-szel, érzékelőkkel felszerelt mobiltelefonnal vagy tablettel használható.

A kiterjesztett valóság virtuális elemeit különböző módon hívhatjuk elő. Vannak olyan alkalmazások, amelyek egy előhívó (marker) segítségével jelenítik meg a virtuális elemeket. Ilyen például, amikor egy meghatározott képet kell az eszközünk kamerájával beolvasnunk a háromdimenziós ábra előhívásához. Ha a képet mozgatjuk, a virtuális alakzatot körbe tudjuk járni. Ilyen alkalmazás például a Quiver vagy az Eddie, amelyek markerként egy-egy nyomtatható ábrát használnak, vagy a Merge Cube, amelynek használatához egy speciális kocka szükséges.

Más esetben a virtuális tartalom előhívásához a helyzetünket (GPS-koordinátát, irányt, gyorsulást) használják. Egy ilyen alkalmazásban adott helyre kell eljutnunk, hogy láthassuk a virtuális tartalmat. Ha a megadott helyen vagyunk, például egy videót, egy zenét vagy egy képet érhetünk el.



► Kiterjesztett valóság: Eddie alkalmazás

A kiterjesztett valóságot sokrétűen használják. Valós térben játszható játékokban virtuális alakzatokat kereshetünk velük; vásárlás előtt kipróbálhatjuk, hogy hogyan mutatna a bútor a szobánkban; múzeumban, városban idegenvezetőként segíthet minket. Emellett természetesen az oktatásban is tudjuk alkalmazni szemléltetőeszközként, a megfelelő helyen megfelelő feladatokat, információkat átadó alkalmazásként. Ilyen például a holokausz emlékeztető alkalmazás. A WallaMe alkalmazás segítségével magunk is elhelyezhetünk virtuális tartalmakat bizonyos helyekre, pontokra.

## Nyelvtanulás

A nyelvtanulás ma már elképzelhetetlen digitális eszközök nélkül. A nyelvtanuláskor szükséges szótárak, fordítóprogramok mobil eszközünkre telepített alkalmazásként állandóan rendelkezésünkre állhatnak. Emellett számos idegen nyelvi oktatóprogramot találhatunk a mobil rendszerek alkalmazásai között.

## Mérések

A mobil eszközöket számos érzékelővel szerelik fel. Ezek az érzékelők, vagy más néven szenzorok sok telefonos funkció működéséhez szükségesek. Ezeket a tanulás során felhasználhatjuk kísérletekhez és mérésekhez. A beépített szenzorok érzékelik a mozgást (pl. gyorsulás, elfordulás), az eszköz helyzetét (GPS-koordináták, iránytű) és a környezet állapotát (nyomás, hőmérséklet, megvilágítás, távolság érzékelése). A szenzorok mérési adatait közvetlenül nem látjuk a mobil eszközünkön, de léteznek olyan alkalmazások, amelyeket feltelepítve kiolvashatjuk, sőt kezelhető formában (általában táblázatként) exportálhatjuk is. Így van mód az adatok elemzésére, kiértékelésére. Ilyen alkalmazás például a Physics Toolbox, amellyel gyorsulás, elfordulás, nyomás, mágnesesség, hangfrekvencia és sok egyéb mennyiség mérhető. A mérési adatok elemzéséhez fizikai, informatikai és matematikai ismeretekre van szükségünk.



► Physics Toolbox mérőeszközei

## Tudományágak, tantárgyak alkalmazásai

Sok alkalmazás elsősorban egy-egy téma vagy tantárgy esetében használható.

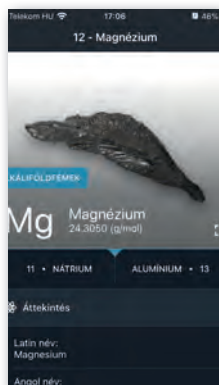
A matematika tanulását segítő szoftverek közül az egyik legalapvetőbb a GeoGebra, amelynek létezik mobilváltozata is. A függvények, geometriai ábrák, szerkesztések, koordináta-geometriai számítások során használhatjuk, de számos egyéb, nem csak szorosan matematikához köthető feladatban is segítségünkre lehet.

Több tantárgyban hasznosak az adattárakat tartalmazó alkalmazások. Ilyenek például a növény- és állathatórózók, a periódusos rendszerek, a történelmi adattárak. Sok olyan alkalmazás van, amely a tananyagot teszi szemléletesebbé: háromdimenziós körbejárható modelleken keresztül mutat be szabad szemmel kevésbé látható vagy távoli dolgokat, történelmi helyszínek rekonstrukcióját.

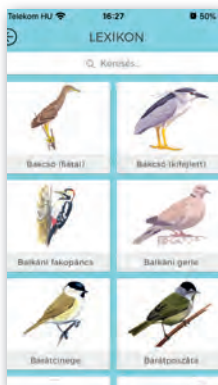




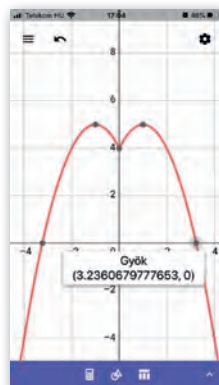
► Fa Book



► Periódusos táblázat 2020



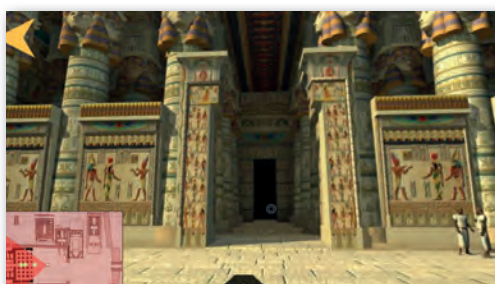
► Madárhatózó



► GeoGebra



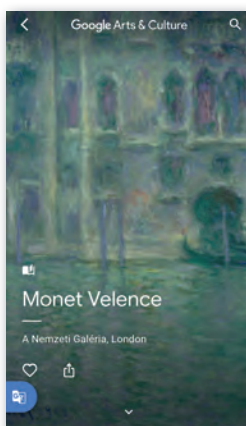
► Virtual Orbitals 3D szemléltető modell



► Ancient Egypt bejárható 3D modell



► Famous composers



► Google Arts&Culture



► Settera

## Kérdések, feladatok

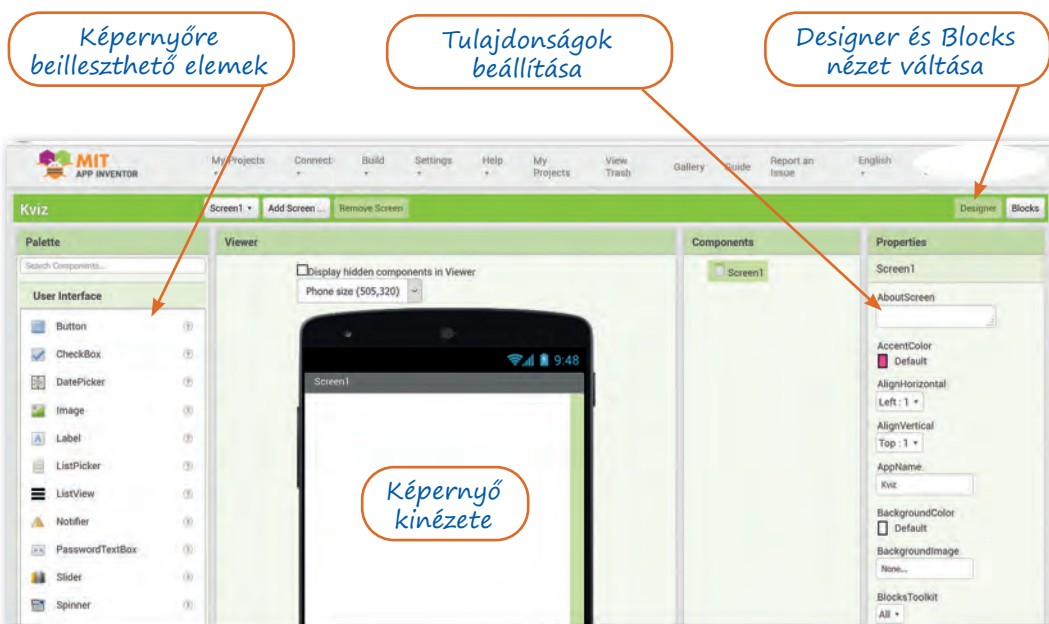
1. Gyűjtsünk alkalmazásokat, amelyekkel egyes tantárgyak tanulását segíthetjük! Próbáljuk ki őket! Indokoljuk meg, hogyan segíthetik a tanulás eredményességét!
2. Soroljuk fel a digitális jegyzetelés előnyeit és hátrányait!
3. Próbáljunk ki mobiltelefonos mérést segítő applikációkat!

## Egyszerű mobilalkalmazás készítése

A mobil eszközök programozása szakértelmet igényel, de vannak olyan eszközök, amelyek ezt egy átlagos felhasználó számára is könnyen érthetővé teszik. Ilyen például az AppInventor alkalmazás, amelynek segítségével egyszerűen készíthetünk programokat Android operációs rendszerre. Az alkalmazásokat online felületen lehet létrehozni. Az ehhez szükséges weboldalt a MIT (Massachusetts Institute of Technology) tartja fenn. Az oldal használatához be kell jelentkezni, amelyhez a Google-fiókunk adatait kell megadnunk. Miután regisztráltunk és elfogadtuk a feltételeket, új projektet indíthatunk.

Az applikációnk elkészítése során két lényegesen különböző felületen fogunk dolgozni. Az első felület a *Designer* ablak: itt állíthatjuk be a képernyő és a képernyőn megjelenő egyes elemek kinézetét. A projekt nevének megadása után ebbe a nézetbe lépünk be. A másik a *Blocks* ablak: ott készíthetjük el az alkalmazásunk kódját. A két nézet között a jobb felső sarokban lévő gombokkal válthatunk.

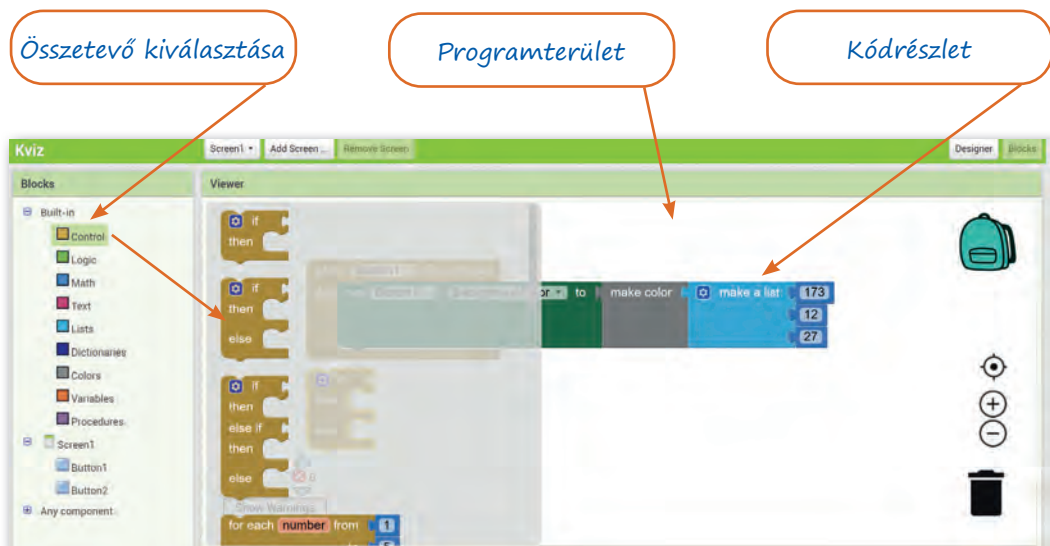
A Designer ablak felépítése:



► App Inventor *Designer* nézete

A középen látható telefon képernyőjén jelennek meg az általunk kiválasztott elemek. A beilleszthető objektumok listája (*Palette*) a telefon képétől balra helyezkedik el, a másik oldalon pedig a tulajdonságait állíthatjuk be. Az elemeket egyszerűen behúzhatjuk a telefon képernyőjére, majd a jobb oldalon beállítjuk a kinézetüket.

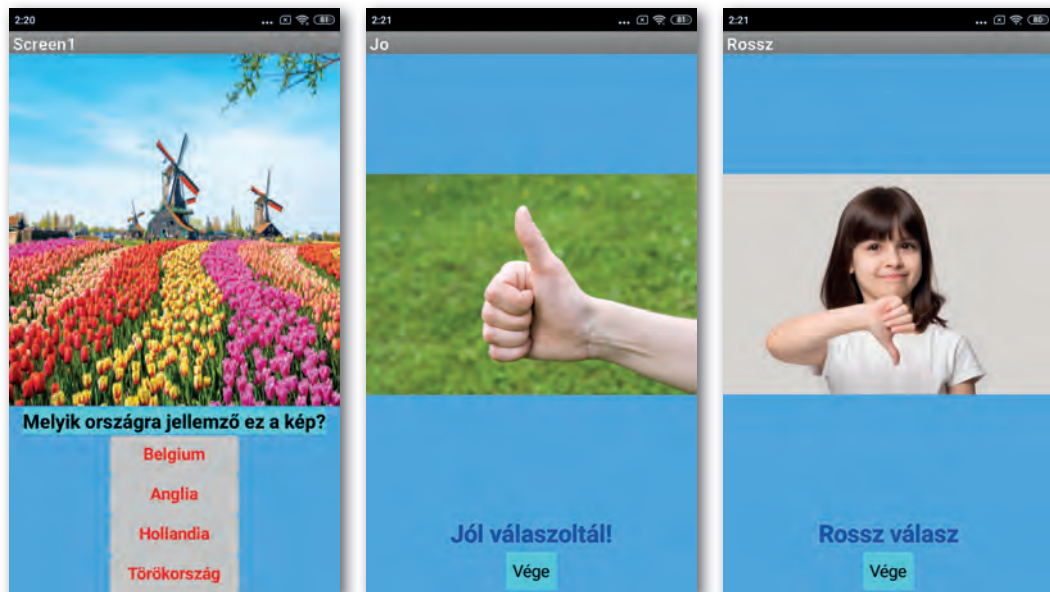
A *Blocks* nézetben a beállított elemeknek megfelelően csoportosítva jelennek meg a program egyes lehetséges utasításai. Ezeket az utasításokat kiválasztva és a kódterületre (*Viewer*) behúzva állíthatjuk össze a programunkat.



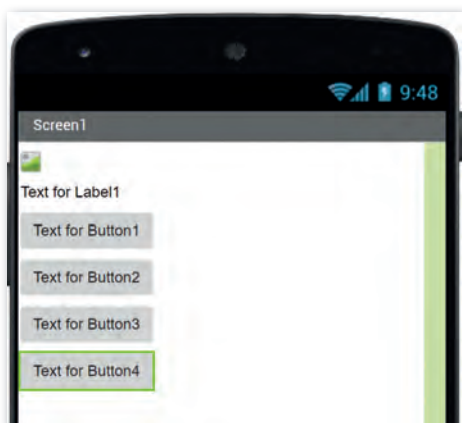
► App Inventor *Blocks* nézete

Próbáljunk meg egy egyszerű programot elkészíteni. A program egy kvízkérdést fog feltenni, amelyre a nyomógombok egyikének megnyomásával válaszol a felhasználó. A válasz helyessége szerint a program megjeleníti a megfelelő képernyőt.

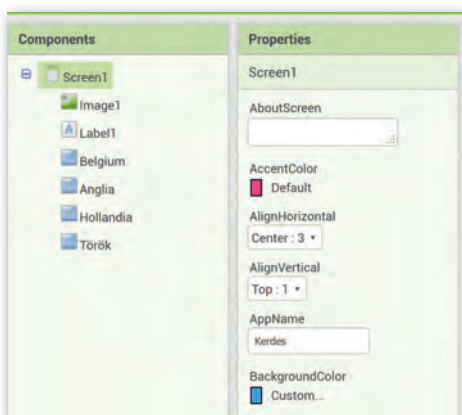
A megvalósítandó terv így néz ki:



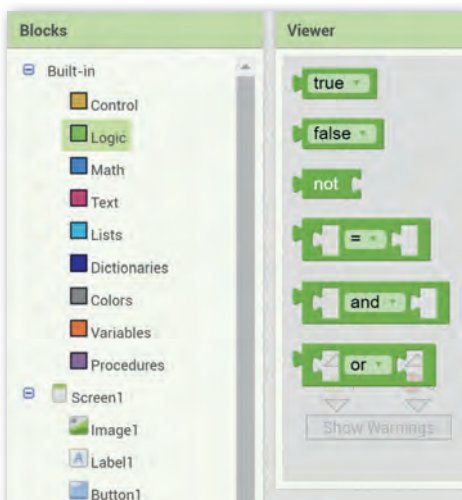
► Képernyőképek



▶ A képernyőn megjelenítendő elemek



▶ Az összetevők tulajdonságainak beállítása



▶ Block nézet: utasítások típusok szerint rendezve

Először hozzuk létre az új projektet a menüsorban a *My Projects > New project* menüpontnál. Legyen a projekt neve *Kerdes*. Ezután a képernyő kinézetét kell felépítenünk a *Designer* nézetben. Az első képernyő kialakításához a bal oldali elemek közül szükségünk lesz egy kép elemre (*Image*), egy szöveges mezőre (*Label*) és négy darab nyomógombra (*Button*). Behúzzuk ezeket a megfelelő sorrendben a képernyőre.

Kialakítjuk a képernyő tartalmát. A jobb oldali részben először az összetevőt kell kiválasztanunk (*Components*), utána a komponens tulajdonságait állítjuk be (*Properties*). Először a kezdőképernyő (*Screen1*) igazítását (*Center*) adjuk meg, és beállítjuk a háttérszínt.

Ezután a kép megjelenítése következik. Ehhez először feltöltjük a megfelelő képet az oldalra. Ezt a *Components* alatt elhelyezkedő *Media* részben tudjuk megtenni. Ha kész a feltöltés, az összetevők között kattintunk az *Image* elemre, majd a jobb oldalon a tulajdonságok között a *Picture* részben kiválasztjuk e feltöltött képet. A kép most megjelenik a képernyőn. Mivel a mérete nem felel meg a képernyőnek, kilóg, ezért ezt is be kell állítanunk. A kép magasságát (*Height*) és szélességét (*Width*) úgy állítjuk be, hogy kitöltse a helyet (*Fill parent*). A szöveges mezőt és a nyomógombokat is beállítjuk. Tetszésünknek megfelelően kiválasztjuk a színeket és a nyomógombok alakját. A nyomógombok feliratát (*Text for Button*) és a képernyőn megjelenő szöveget (*Text for Label*) begépeljük. Ezzel a kezdőképernyő készen van.

Készítünk még egy-egy képernyőt a jó és rossz válaszoknak. Érdekes a két képernyőt megfelelően elneveznünk, és a háttérük színét a kezdőképernyőével azonosra beállítanunk. Mind a kettőn elhelyezzük, beállítjuk a képeket és a szövegeket, nyomógombokat az első képernyőhöz hasonlóan. Ha ezt megtettük, a dizájn elkészítésével készen vagyunk.

Átlépünk a *Blocks* nézetbe. Itt a *Blocks* részben típusok szerint rendezve találjuk meg az algoritmusok szokásos felépítő elemeit és az egyes objektumokhoz tartozó speciális utasításokat.



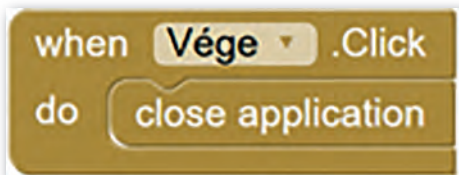
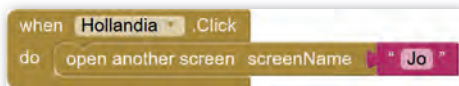
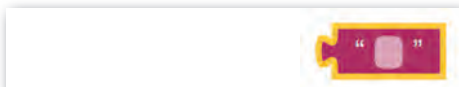
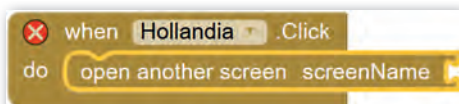
A *Control* csoportban találjuk meg a vezérlőelemeket, elágazásokat, ciklusokat. A *Logic* részben a logikai műveleteket és konstansokat, a *Math* részben a matematikai műveleteket, konstansokat és így tovább. A kategória alsó részén helyezkednek el a *Designer* ablakban beépített elemekhez tartozó műveletek. Az egyes algoritmus elemeket a középső területre húzzuk, és ott megfelelően összeillesztjük.

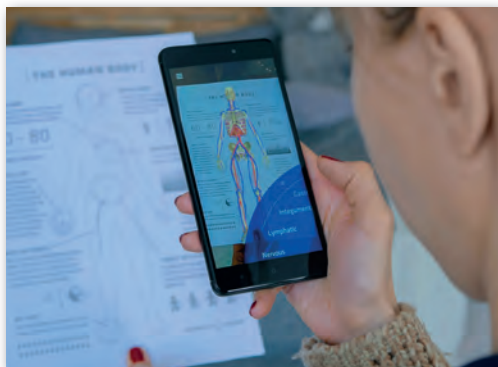
Programunkkal a következőt valósítjuk meg. Ha a felhasználó a helyes válaszra, vagyis a Hollandia feliratú gombra kattint, akkor a jó válasznak megfelelő képernyő jelenjen meg, ha pedig másik nyomógombra, akkor a rossz válasznak megfelelő.

A következőképpen valósítjuk meg az algoritmust: A bal oldali *Blocks* kategóriában Hollandia nyomógombjára kattintunk. Kiválasztjuk azt az elemet, amellyel a gombra való kattintás esetére adunk további utasítást. Ez a képen látható elem lesz. Ezután kiválasztjuk, hogy mi történjen a kattintás hatására. A *Control* csoport elemei között keressük meg a képernyőváltásra vonatkozó blokkelemet (lásd a képen). Az elemek körvonalán megfigyeljük, hogy hol vannak a csatlakozási pontjaik. Ezek segítségével illesztünk hozzájuk új elemet. Összeillesztjük most az előző kettőt. Az így keletkezett elem bal felső sarkában is látszik, hogy az elemmel így még valami gond van. Természetesen az elem másik oldalán is láthatjuk az üres csatlakozási felületet. Az algoritmust végiggondolva tudjuk is a hiányosságot: nem adtuk meg, hogy melyik képernyőt kell betöltenünk. A megadáshoz kiválasztunk egy szöveges konstanst. Ezt a *Text* csoportban találjuk. Az előző elem végére illesztjük, és kitöltjük az új képernyő nevével. Ezzel az elemünk készen lesz, a figyelmeztetés is eltűnik róla. Hasonló módon elkészítjük a többi nyomógombra vonatkozó utasítást, csak ott a másik képernyő betöltését adjuk meg.

A program már működőképes, de gondoskodnunk kell arról, hogy ki lehessen lépni belőle. Ennek érdekében került az értékelés képernyőire a *Vége* feliratú nyomógomb. Átváltva ezekre a képernyőkre, a megfelelő utasítást itt is beállítjuk, ez az applikáció bezárása (*close application*).

Ha mind a két képernyőn beállítottuk ezt a műveletet, készen vagyunk az applikáció összeállításával. A fordítás következik. A menüsorban a *Build* opció választásával a programunkat lefordítjuk. Ezen a ponton választanunk kell, hogy hogyan szeretnénk átvenni a kész programot a mobiltelefonra. A QR-kód lehetőséget választjuk, így a program fordítása





után megjelenő kódot beolvasva le tudjuk tölteni a programot a telefonra. Érdeemes a kipróbáláshoz esetleg egy már használaton kívüli Android operációs rendszerrel felszerelt telefont használni. A letöltött programot telepítjük. Telepítéskor az ismeretlen forrásokat engedélyezzük, a rendszer többször is figyelmeztethet a veszélyekre, de ezt figyelmen kívül hagyhatjuk most. Nincs más hátra, kipróbáljuk a programot, és ha szeretnénk, módosítunk rajta.

### Kérdések, feladatok

1. Készítsünk a fenti leíráshoz hasonló programot!
2. Fejlesszük tovább a programot, készítsünk további kérdéseket, változtassunk a kinézetén, működésén!
3. Alakítsunk három-négy fős csoportokat, és válasszunk az alábbiak közül egy témakört, amelyet közösen részletesebben kidolgozunk!
  - a. Válasszunk egy tantárgyat, amelyhez tanulást segítő applikációkat gyűjtünk! Lássuk el a gyűjteményt leírásokkal, magyarázatokkal!
  - b. Dolgozzunk ki egy mobiltelefon segítségével végzett mérést!
  - c. Készítsünk egy új programot a mobiltelefonra az AppInventor segítségével!



Ebben a fejezetben átismételjük a világhálóval kapcsolatos alapfogalmakat, majd áttekintjük, hogy mi magunk hogyan készíthetünk és publikálhatunk weboldalakat.

## Az internet és a web kapcsolata

Előzetes tanulmányainkból már tudhatjuk, hogy az internet nem más, mint egy globális, az egész bolygónkat behálózó számítógépes hálózat.

Ez a hálózat valójában nem egy fizikai hálózatot jelent, hanem sokféle, önálló hálózatból, illetve hálózatformából áll, amelyek egymással összeköttetésben állnak. A hálózatra csatlakoztatott eszközök egy egyedi hálózati azonosítóval rendelkeznek, amelyet **IP-címnek** (Internet Protocol-címnek) nevezünk. Példa egy ilyen IP-címre: 84.206.104.74.

Az internethálózat sokféle információmegosztási, kommunikációs szolgáltatást támogat. Ilyen például az **e-mail** (elektronikus levél), a **chat** (csevegés) vagy a **fájltviteli szolgáltatás** (pl. FTP, SFTP, SCP), amely az állományok számítógépek közti átvitelét (feltöltését, letöltését) teszi lehetővé.

A **world wide web** (amelyet magyarul **világhálónak** nevezünk) szintén egy olyan szolgáltatás, amelyet az internethálózaton érhetünk el. Tehát a világháló és az internet nem ugyanazt jelentik, bár sokan (tévesen) szinonimaként használják a két fogalmat.

A world wide web megalkotása *Tim Berners-Lee*, valamint *Robert Cailliau* nevéhez fűződik, akik a CERN (Európai Nukleáris Kutatási Szervezet) munkatársai voltak. 1989-ben egy olyan rendszert kezdtek el kidolgozni, amely a világ különböző helyszínein dolgozó kutatók közti információmegosztást támogatta.

1993. április 30-án a CERN bejelentette, hogy a kidolgozott technológiát (www) mindenki szabadon, ingyenesen használhatja. Ebben az évben Magyarországon is elindult az első www-kiszolgáló, ami a `www.fsz.bme.hu` webcímen üzemelt.

1994-ben Tim Berners-Lee megalapította a *World Wide Web Konzorcium* nevű szervezetet (w3.org), amely mind a mai napig koordinálja a nyílt, webes szabványok kidolgozását.



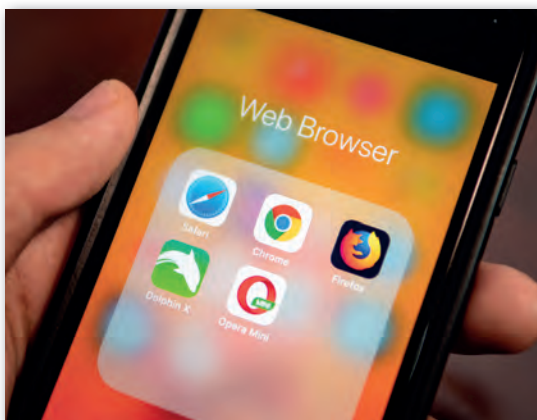
► Sir Timothy John Berners-Lee

## A www (világháló) építőkövei

A világháló egy olyan információs rendszer, amelyre jellemző, hogy **böngészőprogramok** segítségével különböző dokumentumokat (pl. weboldalakat, PDF-dokumentumokat stb.) és egyéb állományokat (pl. képeket, videókat, zenéket stb.) érhetünk el. Ezeket összefoglalóan **erőforrásoknak** (*resource*) nevezzük. Ezen erőforrások között a **hiperhivatkozások** (*linkek*) teremtenek kapcsolatot.

Így egy weboldaltól kiindulva a különböző hivatkozások követésével újabb és újabb oldalakra juthatunk el. Az viszont nem garantálható, hogy a visszafelé irányt is meg tudjuk tenni a hivatkozások segítségével, mivel a **hivatkozások egyirányúak**. A saját weboldalunkról bármilyen weboldalra hivatkozhatunk, de a hivatkozott oldalon ettől még nem biztos, hogy fognak hivatkozni a mi oldalunkra.

A www megalkotása során különböző kérdésekre kellett választ adniuk a tervezőknek. A következőkben ezekkel részletesebben foglalkozunk.



► Böngészőalkalmazások mobil platformon

### Milyen eszközzel érjük el a világhálón publikált anyagokat?

Tudjuk, hogy a világhálón publikált anyagokat jellemzően a **webböngésző programok** segítségével érhetjük el.

Böngészőprogramokat sokféle cég, szervezet fejleszt. A statisztikák szerint a legnépszerűbb webböngészők napjainkban a következők: *Google Chrome*, *Safari* (Apple), *Mozilla Firefox*, *Samsung Internet*, *Microsoft Edge*, *Opera*.

Egy operációs rendszerre akár több, különböző böngészőprogramot is telepíthetünk.

### Feladatok, kérdések

Három-négy fős csoportokban vitassuk meg, hogy mely böngészőprogramokat kedveljük a legjobban, és miért? Állítsunk fel ez alapján népszerűségi rangsort! Ha használunk a felsoroltakon kívül más böngészőprogramokat, azoknak milyen előnyeik, egyedi funkcióik vannak? Minden csoport szóvivője ismertesse az eredményeket!

A böngészőprogram képes megjeleníteni a világhálón közzétett (publikált) tartalmakat, de ehhez meg kell adnunk azt a webcímet, ahol az adott tartalom elérhető. Folytassuk az ismerkedést ezzel!



## Hogyan hivatkozunk az elérendő dokumentumra?

A dokumentumok és más állományok elérési helyére az úgynevezett **URL** (*Uniform Resource Locator*), vagyis *Egységes Erőforrás Helymeghatározó* segítségével hivatkozhatunk. Az alábbiakban néhány példát látunk erre.

<a href="https://www.nava.hu/">https://www.nava.hu/</a>	A Nemzeti Audiovizuális Archívum weboldala.
<a href="https://www.mnm.hu/kiallitasok">https://www.mnm.hu/kiallitasok</a>	A Magyar Nemzeti Múzeum portál egy aloldala, amely a kiállításokat mutatja be.
<a href="http://www.fortepan.hu/_photo/display/154350.jpg">http://www.fortepan.hu/_photo/display/154350.jpg</a>	A Fortepan közösségi fotóarchívum weboldalán elérhető egyik kép webcíme.
<a href="https://www.w3.org/standards/">https://www.w3.org/standards/</a>	Webes szabványok a W3 konzorcium weboldalán.
<a href="https://hirmagazin.sulinet.hu/">https://hirmagazin.sulinet.hu/</a>	A Sulinet hírmagazin webportálja.
<a href="https://www.google.com/search?q=világháló">https://www.google.com/search?q=világháló</a>	A Google kereső találati oldala a világháló szóra keresve.
<a href="https://hu.wikipedia.org/wiki/Tim_Berners-Lee#Kitüntetései">https://hu.wikipedia.org/wiki/Tim_Berners-Lee#Kitüntetései</a>	A Tim Berners-Leeről szóló Wikipédia-oldal kitüntetésekkel foglalkozó oldalrészére mutató webcím.
<a href="file:///C:/honlapra/hobbi.html">file:///C:/honlapra/hobbi.html</a>	A saját számítógépünk c:\honlapra mappájában található <code>hobbi.html</code> nevű dokumentum.

Ahogy a példákban is látszik, az URL több részre tagolódik. Ezek a következők:

### 1. Séma

A séma utal arra, hogy milyen szabályrendszer (protokoll) szerint történik a kommunikáció a hálózat tagjai között. A protokoll leírja, hogy hogyan kell történnie a kapcsolatfelvételnek, hogyan történik az üzenetváltás, mi az üzenetek felépítése és az egyes üzenetekben milyen adatok lehetnek.



**Érdekesség:** A protokoll kifejezést hétköznapi értelemben is használjuk, a különböző viselkedési szabályokra, illetlani ismeretekre utalva.

A webes dokumentumok, állományok vonatkozásában leggyakrabban a `http` vagy `https` sémát szoktuk megadni. A séma neve után egy kettőspontot kell írni. A kettőspontot webes tartalmak esetén két perjel (`//`) követi.

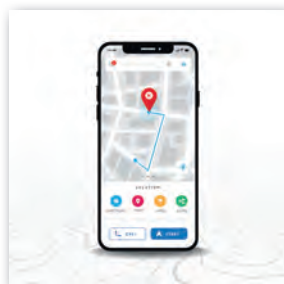
*Fontos! Amikor a böngészőprogramban megadunk egy webcímet, sokszor nem szoktuk begépelni az előtagot (pl. `http://`, `https://`), mégis megjelenik a weboldal. Ennek oka, hogy a böngészőprogramok ilyen esetekben alapértelmezetten a `http`-sémát használják.*

Amennyiben a saját számítógépünk egy mappájából nyitunk meg egy dokumentumot (pl. weblapot) a webböngészőben, akkor láthatjuk, hogy nem a `http` sémát, hanem a `file` sémát használja a böngészőprogram.

## 2. Útvonal

A séma után kell megadnunk az erőforrás elérési útvonalát. De ezt hogyan tudjuk megtenni?

*Ha egy épülethez szeretnénk eljutni, akkor használhatjuk a helyszín GPS-koordinátáit. Például a Parlament épületének a GPS-koordinátái: N47.50708 E19.04591. Ez a számsor jól használható egy navigációs szoftverben, de nehezen lehetne megjegyezni. Helyette így is megadhatnánk ugyanezt a címet: Budapest V. kerület, Kossuth tér 1–3. Ez utóbbi cím szintén azonosítja a helyet, és jól megjegyezhető.*



Láttuk korábban, hogy az internethálózatra csatlakoztatott eszközök egy egyedi IP-címmel rendelkeznek (pl. 84 . 206 . 104 . 74). A webböngésző programban akár ezen IP-cím segítségével is elérhetnénk a gépek szolgáltatásait. Ez azonban túlságosan bonyolult lenne, hiszen éppúgy egy hosszú, számokból álló azonosítót kellene használnunk, mint az előbbi példában a GPS-koordinátákat. Másrészt ezek az IP-címek változhatnak. Elképzelhető, hogy egy eszköz minden csatlakozáskor más-más IP-címet kap.

Ezért fontos, hogy a webes kiszolgálókra ne csak IP-cím alapján, hanem egy könnyebben megjegyezhető, állandó névvel is hivatkozassunk. Ilyen a korábbi példánkban a város, utca, házszám. Ez az azonosító az **állomásnév**, vagy más néven **hosztnév** (hostname). A teljes elérési útvonalnak azonban nemcsak az állomásnév a része, hanem az úgynevezett doménnev is.

A **doménnev** (domain name) egy olyan egyedi azonosító, amely kettő vagy több részből áll, és ezeket pontok választják el. A doménnev végződése (a korábbi példákban a *hu*, *com* és *org*) a **legfelső szintű tartományt** jelenti. Ezek lehetnek országokra utaló, illetve kategóriákra utaló rövidítések. Például a *hu* kód Magyarországot jelöli, a *jp* Japánt, az *org* a szervezeteket, a *com* az üzleti weboldalakat, és így tovább.

A legfelső szintű tartomány előtt a **második szintű tartományt** találjuk. Például a *sulinet.hu* tartománynévben a *sulinet* a második szintű tartománynév.

Második szintű tartománynevet meghatározott díjért bárki lefoglalhat saját célra (pl. vállalkozása, alapítványa, családja számára), amennyiben az adott név még szabad.

**Tipp!** Azt, hogy egy doménnev lefoglalható-e még, az Internet Szolgáltatók Tanácsa weboldalán (<http://www.domain.hu/domain/szabad-e/>) lehet ellenőrizni.

### Feladatok, kérdések

Nézzük meg, hogy a saját családnevünkkel megegyező doménnev lefoglalható-e (pl. Balaton Éva esetén azt kell ellenőrizni, hogy a *balaton.hu* doménnev szabad-e).

Mit tapasztalunk? Az osztály mekkora hányada tudná regisztrálni a saját nevét?

A második szintű tartomány tulajdonosa saját hatáskörben akár további aldoménneveket (harmadik, negyedik szintű) is kioszthat, és **helyi állomásneveket (hosztnév)** is meghatározhat. Az állomásnév egyben doménnev is lehet. Például a *hirmagazin.sulinet.hu* cím esetén a doménnev a helyi állomásnévből (*hirmagazin*) és a *sulinet.hu* szülődoménnévből áll össze.

## Hivatkozás mappákra és a bennük található erőforrásokra

A webes kiszolgálókon (szervereken) az információk mappaszerűen vannak szervezve, kicsit hasonlóan ahhoz, mint a saját számítógépünkön is. Az elérési út azt írja le, hogy milyen útvonalon érhető el az adott erőforrás.

*Például a [http://www.fortepan.hu/\\_photo/display/154350.jpg](http://www.fortepan.hu/_photo/display/154350.jpg) webcím esetén a [www.fortepan.hu](http://www.fortepan.hu) tartományhoz tartozó szerver `_photo` mappájában van egy `display` mappa, amelyben a `154350.jpg` nevű képre hivatkozunk.*

Ha nem adunk meg útvonalat, vagy csak egy mappa nevét adjuk meg, akkor egy alapértelmezett tartalom fog megjelenni a kiszolgáló szerver beállításai alapján (pl. `index.html`, `index.php`, `default.aspx`)

### Feladatok, kérdések

1. Szerveződjünk három-négy fős csoportokba! Egy böngészőprogramban nyissuk meg a saját iskolánk weboldalát! A böngészés során próbáljuk kideríteni, hogy vajon melyik aloldalnak van a leghosszabb webcíme! Osszuk fel egymás között, hogy ki melyik részét térképezi fel a weboldalnak, például a menüpontok alapján. Jegyezzük fel a leghosszabb webcímet, és értelmezzük, hogy az elérési útvonalban mi mit jelenthet!
2. Vannak-e az iskolának harmadik szintű tartománynevei? Ha igen, akkor melyek ezek? Gyűjtünk össze párat!

### 3. Lekérdezési paraméterek

A webcímekben akár paraméterek is lehetnek, amelyek alapján a weboldal akár más-más tartalmat tud szolgáltatni. A paramétereket *kulcs = érték* formában lehet megadni, az első kulcs elé pedig kérdőjelet kell tenni.

*Például a <https://www.google.com/search?q=világháló> webcímében paraméterként egy *q* nevű kulcs van megadva, amelynek értéke a világháló szöveg. A *q* az angol *query* (lekérdezés) szó rövidítése. Ezt a kulcsot és értéket egy program feldolgozza, és eredményként azon weboldalak listáját mutatja meg, amelyekben szerepel a „világháló” szöveg.*

Ha több ilyen kulcs lenne, akkor azokat `&` jellel kellene elválasztani.

*Például a <https://www.google.com/search?q=világháló&num=3> webcím már két kulcsot is tartalmaz. A második a *num*, ami az angol *number* (szám) szó rövidítése. Ez a megjelenítendő találatok számát jelenti.*

### Feladatok

Próbáljuk ki a Google keresőjében azt, hogy közvetlenül a webcímekben módosítjuk a paraméterek értékét! Keressünk rá ezzel a módszerrel a „Duna” kifejezésre!

### 4. Oldalrész

A webcímek végén egy olyan azonosító is állhat a `#` jel után, amely egy oldalrészre hivatkozik. Ilyenkor a böngészőprogram az oldal adott elnevezésű részéhez fog ugrani.

**Fontos!** Csak akkor tudunk egy weboldal adott részére ugrani, ha az oldal készítője az adott oldalrészét ellátta egy egyedi névvel. Ezalól kivételt jelent az oldal teteje, amelyre név nélkül, csak a `#` jel megadásával hivatkozhatunk.

Például a [https://hu.wikipedia.org/wiki/Tim\\_Berners-Lee#Kitüntetései](https://hu.wikipedia.org/wiki/Tim_Berners-Lee#Kitüntetései) webcím a Tim Berners-Leeről szóló Wikipédia-oldal azon részére ugrik, ahol a kapott kitüntetéséről van szó.

## Feladatok

A Wikipédia oldalán keressünk olyan szócikket, ami egy olyan témával kapcsolatos, amelyet mostanában tanultunk, és rendelkezik oldalrész-hivatkozással. Másoljuk ki az oldalrész-hivatkozást tartalmazó linket, és mentjük el egy állományba!

## Az URN és URI fogalma

**Érdekesség.** Nemcsak az elérési útvonal, hanem speciális esetben egy **név (azonosító) alapján is hivatkozhatunk** egyes objektumokra. Ezt az URN (*Uniform Resource Name*), vagyis *Egységes Erőforrás Név* segítségével tudjuk megtenni. Ebben az esetben viszont alapvető követelmény, hogy a név világviszonylatban (térben és időben) egyedi legyen, vagyis valóban egyértelműen azonosítson egy elemet.

A saját nevünk például nem lehet URN, mert nem garantálható, hogy egy másik embert (akár a világ másik részén) ne hívjanak ugyanígy, most, vagy a jövőben. De az is lehet, hogy a múltban is élt már ilyen névvel valaki.

De akkor mi lehet URN? Bizonyára találkozta már az ISBN számmal a könyvek borítóin. Ez az iktatószám világszinten egyedi. Magyarországon ezen azonosítókat a kiadók kérésére az Országos Széchényi Könyvtáron belül működő Magyar ISBN és ISMN Iroda osztja ki. Például Magyarország Alaptörvényére ezzel az URN azonosítóval hivatkozhatunk: **urn:isbn:9786155269813**

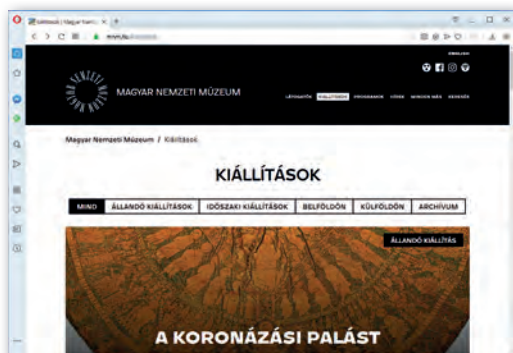
Mind a korábban látott **URL-ek**, mind az **URN-ek** beletartoznak egy **nagyobb kategóriába**, amely az **URI** (*Uniform Resource Identifier*) névre hallgat. Ezt magyarul *Egységes Erőforrás Azonosító*nak hívjuk. Vagyis minden URL egyben URI is, de nem minden URI-ra igaz, hogy URL.



## Hogyan történik a kommunikáció?

A világhálón jellemzően a **http** (*hypertext transfer protocol*, magyarul *hiperszöveg-átviteli protokoll*) szerint történik a kommunikáció, illetve biztonsági okokból gyakran a **https**-sémát használjuk a webböngészőkben, ami azt jelenti, hogy titkosított, biztonságos formában történik a kommunikáció.

A kommunikáció kérés-válasz formájában zajlik a kliens és a szerver között. A kliens (ami tipikusan a webböngésző program, de más alkalmazás is lehet) kérést küld a kiszolgálónak, vagyis a szervernek. Erre a kérésre válaszol a webszerver.



► Weboldal megjelenése egy böngészőben

A kommunikáció során a webböngésző megkapja az adott weboldal forráskódját is. A weboldal tartalmazhat képeket, videókat és egyéb elemeket is, amelyek szükségesek a megjelenítéshez. Ezeket a böngészőprogram szintén le kell, hogy kérje a szervertől újabb kérés-válasz ciklusokban, majd a kapott adatok alapján megjeleníti az oldalt.

## Hogyan, milyen nyelven írjuk le a dokumentumok tartalmát, illetve kinézetét?

### A tartalom leírása (HTML)

A világhálón jellemzően a HTML (*HyperText Markup Language*) je-  
lölőnyelvet használjuk arra, hogy a weboldalak tartalmát leírjuk. Ennek a nyelvnek több verziója is van. A legutolsó és egyben legkor-  
szerűbb változat a HTML5. Ezzel fogunk mi is foglalkozni.

**Fontos!** A HTML nem programozási nyelv, hanem a webes doku-  
mentumok tartalmának leírására szolgáló leírónyelv.

A HTML nyelvben úgynevezett tagek (kiejtve tegek), vagy magyarul címkék segítségével tudjuk leírni a dokumentum tartalmát. A címkék „<” jellel kezdődnek és „>” jellel végződ-  
nek. Például egy bekezdés kezdetét a <p> taggel jelöljük. A p az angol *paragraph*, vagyis *paragrafus* szó rövidítése.

Egy címke hatása addig tart, amíg meg nem adjuk a záró párját. A záró címke hasonló,  
mint a kezdő címke, csak egy „/” karakter van a címke neve előtt. Vagyis egy bekezdést így  
jelölhetünk a HTML nyelv segítségével:

```
<p>Ez egy bekezdés</p>
```

Vannak olyan címkék is, amelyeknek nincsen záró párjuk. Ilyen például a <br> tag is,  
amellyel az utána következő szöveget új sorba törhetjük.

```
<p>Ez egy bekezdés, <br> amiben sortörést alkalmaztunk.</p>
```

A címkék használata során akár paramétereket is megadhatunk, sőt bizonyos esetekben  
ez kötelező is. A paramétereket a címke nyitó részében kell megadnunk paraméter="ér-  
ték" alakban. Egy címkéhez több paramétert is írhatunk, ekkor szóközzel kell elválaszta-  
nunk őket. Vannak úgynevezett globális paraméterek. Ezeket bármilyen tag esetén megad-  
hatjuk. Ilyen például a title, class, id. Más paramétereket csak  
az adott tag esetén használhatunk.

Például egy kép beillesztéséhez az <img> címkét kell használnunk.  
Az img az angol image (kép) szó rövidítése. Viszont ebben az esetben  
kötelezően meg kell adnunk azt is, hogy hol érhető el a kép (mi az URL-  
je). Ez kerül az src paraméterbe. Az src a source (forrás) szó rövidítése.  
Sőt, azt is le kell írunk szövegesen, hogy mi látható a képen. Ez az alt  
paraméterbe kerül. Az alt az alternate (helyettesítő leírás) rövidítése..

```

```

Az előbbi példában a goldi.jpg képet illesztettük be az oldalra. Mivel nincs megadva sem-  
milyen hosszabb útvonal, csak a fájl neve, ez azt jelenti, hogy a kép ugyanabban a mappában  
van, mint maga a weblap, amelybe be van illesztve. Ha esetleg a képet nem tudná letölteni a  
böngészőprogram, akkor helyette az alt paraméterben megadott leíró szöveget jeleníti meg.  
A vak emberek szintén ezen szöveg alapján tudják eldönteni, hogy mi látható a képen.



## A kinézet leírása (Stíluslap)

A HTML5 nyelv segítségével a weboldalak tartalmát írhatjuk le, a kinézetét viszont nem. Erre a CSS (*Cascading Style Sheets*) szabvány szolgál, amelyet magyarul legtöbbször *lépcsőzetes stíluslapoknak*, *csatolt stíluslapoknak* vagy *egymásba ágyazott stíluslapoknak* fordítanak.

A szabványnak több verziója van, a jelenlegi legkorszerűbb változat a CSS3.

A stíluslapok szakszerű alkalmazásának számos előnye van:

- Számos formázást (pl. színbeállítások, szövegbeállítások) és oldalfelépítést (pl. két-oszlopos elrendezés) támogat.
- A weboldalak karbantartása egyszerűsödik, mivel az oldal kinézetének leírása nem keveredik a tartalom leírásával.
- Segíti az akadálymentes oldalak előállítását (később erre még kitérünk).
- Különböző eszközökre (pl. okostelefonokra, nyomtatókra, okosórákra) külön-külön stíluslap készíthető, amely biztosítja az optimális elrendezést és használhatóságot.

Nézzünk egy egyszerű példát! Korábban láttuk, hogy a bekezdést a `<p>` címkével tudjuk jelölni. Hogyan tudnánk minden bekezdés kinézetét megváltoztatni a stíluslap segítségével úgy, hogy a szöveg kék színű legyen, és nagyobb betűmérettel jelenjen meg?

Ahhoz, hogy **kijelöljük** a formázáshoz az oldal összes bekezdését, először meg kell adnunk a tag (címke) nevét. Ezért ezt a részt **kijelölőnek** vagy **szelektornak** hívjuk.

```
p {  
    color: blue;  
    font-size: 120%;  
}
```

A szelektor után a `{` és `}` jelek közötti **deklarációs blokkban** megadhatjuk, hogy hogyan szeretnénk megjeleníteni az adott elemet. Itt meg kell adnunk a tulajdonságokat, illetve az ahhoz tartozó értékeket.

A tulajdonság-érték megadást **deklarációnak** nevezzük. A különböző deklarációkat pontosvesszővel kell elválasztani egymástól.

Például a `color` tulajdonság a szöveg színét állítja be. Értékként megadhatunk különböző színneveket angolul.

A `font-size` tulajdonság a betű méretét jelöli. A 120% érték azt jelenti, hogy az alapértelmezett méret 120%-át szeretnénk beállítani, vagyis kis mértékben meg szeretnénk növelni a betűméretet.

## CSS osztályok használata

Persze egyáltalán nem biztos, hogy minden egyes bekezdést ugyanúgy szeretnénk formázni. Ebben az esetben használhatunk **osztályokat**. A HTML-kódban az egyes elemeket osztályokba sorolhatjuk. Ehhez a `class` paramétert kell használnunk. Az osztály nevét mi találhatjuk ki. A névben számok, betűk, kötőjelek és aláhúzások lehetnek, de például szóköz és más speciális karakter nem!



Ha például ki akarunk emelni egy bekezdést a többi közül, akkor használhatjuk osztálynévként a `kiemelt` szót!

**Tipp!** Mindig próbáljunk meg olyan osztálynevet megadni, amely általánosan írja le a formázás jellegét. Például ne adjunk olyan osztálynevet, hogy *piros*, mert lehet, hogy később az elem kinézetét megváltoztatjuk lilára, és akkor ez az osztálynév félrevezető lesz.

A CSS-kód:

A HTML-kód

```
p.kiemelt {           <p class="kiemelt">Ez egy kiemelt bekezdés</p>
  color: blue;
  font-size:120%;    <p>Ez pedig egy normál bekezdés</p>
}
```

Látható, hogy a CSS-kódban pontosan ugyanazt a nevet kell használnunk, mint a HTML-kódban, úgy, hogy a bekezdést jelölő `p` után egy pontot teszünk, majd leírjuk az osztály nevét. Ebben az esetben a `kiemelt` nevű osztály csak a bekezdés esetén érvényesül.

Létrehozhatunk olyan általános osztályokat is, amelyeket akár minden elemnél felhasználhatunk. Ekkor a pont elé ne írjunk semmit!

A CSS-kód:

A HTML-kód

```
.szegely {           <p class="szegely">Szegélyezett
  border:1px solid blue; bekezdés</p>
}                   <h1 class="szegely">Szegélyezett
                  címsor</h1>
```

A fenti példában a `szegely` osztályban azt adtuk meg, hogy az elem körül legyen egy szegély (*border*), ami 1 képpont vastag (*1px*), folytonos vonallal van megrajzolva (*solid*), kék (*blue*) színnel. Mivel ez egy általános osztály, több tag esetén is érvényre jut a formázás. Használhatjuk bekezdésnél (`<p>`), címsornál (`<h1>`), és még sok más elemnél is.



## Reszponzív weboldalak



A weboldalakat napjainkban igen sokféle eszközön tekintjük meg. Használhatunk asztali számítógépeket extranagy felbontású monitorokkal, notebookokat nagy felbontású kijelzőkkel, közepes kijelzőjű tableteket és kis kijelzős okostelefonokat, az egészen apró kijelzőjű okosórákról nem is beszélve!

Bármelyik eszközzel is nézzük meg a weboldalt, nagyon fontos, hogy az oldal az adott kijelzőn optimális módon jelenjen meg, vagyis egyszerűen lehessen navigálni az oldalak között, jól olvasható legyen a tartalom, ne kelljen átméretezni, illetve indokolatlanul sokszor görgetni a tartalmat. Ha egy weboldal teljesíti ezeket a követelményeket, akkor **reszponzív kialakítású weboldalnak** nevezzük.

Amikor magunk készítünk weboldalakat, akkor is törekednünk kell arra, hogy az oldal reszponzív legyen. Később látjuk majd, hogy akár különböző sablonokból is kiindulhatunk a weblapok készítése során. Itt mindig próbáljunk olyan sablont, illetve stílust kiválasztani, amely reszponzív megjelenést tesz lehetővé!

Azt, hogy egy weboldal reszponzív-e, úgy tesztelhetjük le, hogy többféle eszközön is megtekintjük az oldalt, illetve például asztali számítógépen a böngészőablakot átméretezzük (lecsökkentjük, felnagyítjuk).

### Feladatok, kérdések

Keressünk a saját hobbinkhoz kapcsolódó weboldalt, és vizsgáljuk meg, hogy az reszponzív módon lett megvalósítva, vagy sem. Használjunk a teszteléshez okostelefont, notebookot és/vagy asztali számítógépet!

Amikor mindenki elkészült, osszesítsük az eredményeket! A vizsgált honlapok közül mekkora hányad volt reszponzív megvalósítású?



## Weboldalak akadálymentessége

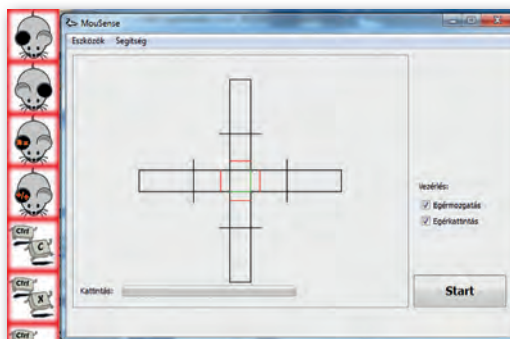
Nagyon sok, fogyatékkal élő embertársunk is böngészik a világhálón, ezért rendkívül fontos, hogy a weboldalakat olyan formában készítsük el, hogy mindenki számára, **akadálymentes** módon használhatók legyenek. Ha figyelünk a honlapok akadálymentes megvalósításra, az minden felhasználó számára előnyökkel járhat.



A fogyatékkal élő felhasználók jelentős hányadára jellemző, hogy a számítógép kezeléséhez, illetve a böngészőprogramok használatához valamilyen segédeszközt használnak. Ez lehet például egy speciális billentyűzet, olyan szájba vehető vagy fejre erősíthető pálca, amellyel le lehet nyomni a billentyűket, lehet Braille-kijelző a vak emberek által használt pontírás megjelenítésére, vagy akár egy speciális segédprogram (pl. képernyőolvasó program, fejegér, képernyőnagyító program) is.



▶ Braille-írás (pontírás) megjelenítésére képes kijelző a vak felhasználók számára



▶ Fejegér-alkalmazás (Mousense), amely lehetővé teszi, hogy az egeret ne kézzel, hanem fejmozgással irányítsák a felhasználók

A **vak emberek** nagyon gyakran úgynevezett képernyőolvasó programot használnak, amely felolvassa számukra a böngészőprogramban megnyitott weboldalak tartalmát, mégpedig a weblap forráskódja alapján. Ebben az esetben például nagyon fontos, hogy a **képek és más médiaelemek (pl. animációk, videók)** tartalmát **szövegesen** is leírjuk. Használjunk egyértelmű, jól érthető, a tartalomhoz megfelelően kapcsolódó **címsorokat**. Szintén alapvető, hogy a tartalom leírásakor mindig a megfelelő elemeket használjuk. Egy címsor nem attól lesz címsor, hogy nagy, félkövér betűkkel jelenítjük meg, hanem hogy a megfelelő HTML-címkét használjuk a kódban. Fontos, hogy lehetőleg **ne állítsunk be háttérzénét** az oldalon, mivel ez zavaró a vak emberek számára (illetve nem csak számukra). A linkek szövegét úgy kell megfogalmaznunk, hogy abból kiderüljön, hogy milyen információt találunk rajta. Például, ha a „*Kattints ide*” szöveget állítjuk be hivatkozásként, akkor ebből nem derül ki, hogy mi fog történni a linkre kattintáskor.

A **gyengénlátó** emberek számára biztosítani kell, hogy jól olvasható, kontrasztos legyen a weboldal, és akkor is jól jelenjen meg, ha a betűméretet megnövelik a böngészőprogramban. Az animációk elkészítésénél is ügyeljünk a kontrasztarányra, illetve adjunk lehetőséget az animáció megállítására, kimerevítésére.

A **siket emberek** számára nagyon fontos, hogy a témát szemléletesen mutassuk be. Amennyiben meg tudjuk oldani, a videókat feliratozzuk, vagy a videók felhasználásakor részesítsük előnyben azokat, amelyek felirattal vannak ellátva. Az akadálymentesség egy mélyebb szintjét valósítják meg azok a videók, ahol jelnyelvi tolmácsolás is látható. A vak, illetve siket felhasználók számára a videó teljes szövegű átiratának megléte is nagyon fontos lehet. Ez az állomány tartalmazza a videó minden fontos történésének (párbeszédek, cselekmények) szöveges leírását, hasonlóan, mintha egy forgatókönyvet írnánk.

Nagyon sok **színtévesztő**, illetve **színvak** felhasználó nehézségekbe ütközik akkor, ha bizonyos információkat csak színekkel különböztetünk meg egymástól. Az alábbiakban egy olyan példát láthatunk, amikor a színnel való megkülönböztetés nem elegendő. Ilyen esetben például használhatunk ikonokat is a helyes, illetve helytelen válaszok jelölésére.

#### Minek a rövidítése a HTML?

- HyperText Markup Language
- HyperText Meta-Language
- HiperText Meta-Language
- HiperText Markup Line

► Teszt, melyben zöld és piros színekkel van megkülönböztetve a helyes és helytelen válasz

#### Minek a rövidítése a HTML?

- HyperText Markup Language
- HyperText Meta-Language
- HiperText Meta-Language
- HiperText Markup Line

► Ugyanez a teszt egy olyan ember szemével, aki zöld-sárga-piros tartományba eső színeket nem tudja megkülönböztetni

A **mozgáskorlátozott** emberek egy része nehézségekbe ütközik, ha billentyűzetet kell használnia, vagy az egérrel finom mozdulatokat kell elvégeznie. Ezért elegendő időt kell hagyni az adatbevitelre, illetve ügyelni kell arra, hogy ne kelljen nagyon kicsi területre kattintaniuk a felhasználóknak. Figyelni kell arra, hogy a weboldal csak billentyűzettel (egér nélkül) is használható legyen, de ez például a vak felhasználók számára is rendkívül fontos.

### Feladatok, kérdések

1. A *funkify.org* webcímen találunk egy ingyenesen kipróbálható szimulátort, amelyet a Google Chrome böngészőbe lehet telepíteni kiterjesztésként. Ez a szimulátor különböző állapotokat (pl. gyengénlátás, színtévesztés, remegő kezek) tesz kipróbálhatóvá. Próbáljuk ki ezeket a funkciókat úgy, hogy közben a saját iskolánk weboldala van megnyitva a böngészőprogramban!
2. Akár átmenetileg is kerülhetnek a felhasználók olyan helyzetbe, mint ha fogyatékkal élnek. Például ha rossz a hangkártya a számítógépben, akkor nem fogjuk hallani a számítógép hangját, akárcsak a siket emberek. Szerveződjünk három-négy fős csoportokba, és vitassuk meg, hogy milyen hétköznapi helyzetekben fordulhat elő az, hogy átmenetileg hasonló problémába ütközünk, mint a fogyatékosokkal élő embertársaink!

## Készítsünk weblapot!

A weboldalak készítésének több módja is van. Kisebb, pár oldalból álló honlapokat elkészíthetünk úgy, hogy a HTML-állományok forrását egy kódszerkesztőben készítjük el, vagy használhatunk segédprogramokat, amelyek hasonlítanak ahhoz, mintha egy szövegszerkesztő programban dolgoznánk, de az eredményt képesek elmenteni HTML-formátumban. Amennyiben a kódot közvetlenül szerkesztjük, majd azt publikáljuk, akkor valójában **statikus oldalakat** készítettünk.

### Dinamikus honlapok készítése

Összetettebb weboldalakat, amelyek számos oldalból állnak, és esetenként többben is szerkesztik a tartalmukat, már nem hatékony statikusan előállítani. Érdemesebb egy **CMS** rendszert (*Content Management System*), vagyis *Tartalomkezelő rendszert* használni, amely az oldalakat valós időben, **dinamikusan** állítja elő. A dinamikusság azt jelenti, hogy a webszerveren futó alkalmazás állítja elő azt a HTML-kódot, amelyet majd a böngésző megjelenít. Így például megtehetjük azt is, hogy egy közösségi oldalon megjelenő bejegyzés automatikusan megjelenik a weboldalon is anélkül, hogy nekünk kellene a tartalmat módosítanunk.



### Statikus honlapok készítése

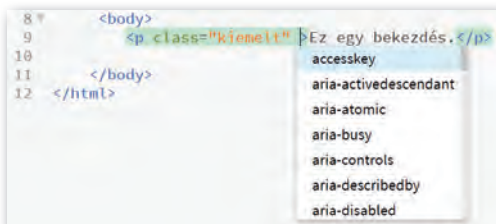
Későbbi tanulmányainkban több tartalomkezelő rendszer használatával is megismerkedünk. Azonban ezekben a rendszerekben is előnyös, ha a HTML-kódot közvetlenül tudjuk módosítani, illetve a kinézetet leíró stíluslapállományt testre tudjuk szabni. Ezért először ismerkedjünk meg a statikus oldalak elkészítésének módjával!

Ahhoz, hogy egy statikus honlapot elkészítsünk, egy nagyon egyszerű kódszerkesztő programot is használhatunk.

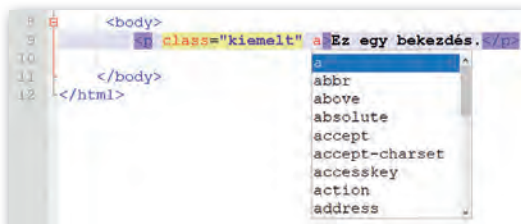
Előnyös, ha ez a program rendelkezik a következő tulajdonságokkal:

- Kijelzi a sorok sorszámát, mert így egyszerűbben meg tudjuk találni az adott sorszámu sort, ha például egy tanári bemutatót követünk.
- Képes színnel megkülönböztetni a tageket, a paramétereket és a paraméterek értékeit. Ezt szintaxiskiemelő funkciónak nevezzük.
- Amikor elkezdjük gépelni a tageket vagy paramétereket, akkor felkínálja, hogy milyen tagek/paraméterek kezdődnek az adott karakterekkel, így nem kell feltétlenül pontosan emlékeznünk az adott kifejezésre, hanem akár ki is választhatjuk a megjelenő listából.

## Kódszerkesztő alkalmazások



- Kódkiemelés és paraméterek felkínálása a Brackets szerkesztőprogramban



- Kódkiemelés és paraméterek felkínálása a Notepad++ szerkesztőprogramban

Ha kiválasztottuk a megfelelő programot, amelyben kódolni fogunk, akkor induljunk ki az alábbi HTML5-sablonból! Ebben fogjuk elhelyezni az oldal tartalmát, a különböző HTML-címkeket használva.

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
  </body>
</html>
```

- HTML5-alapstruktúra

Az első sor tartalmazza a **dokumentum-típust**. Ez azt írja le, hogy melyik szabvány szerint írjuk le a tartalmat, Ez a konkrét sor most arra utal, hogy a HTML5-szabvány szerint készítettük el az oldalt. Ez a sor nagyon fontos. Ha hiányzik, akkor a böngészőprogram nem biztos, hogy helyesen fogja megjeleníteni az oldalt.

A következő sorban a `<html>` taget helyezzük el, amely jelzi a böngészőnek, hogy egy HTML-dokumentumról van szó. A `lang` paraméterben az oldal nyelvét kell megadnunk. Magyar nyelvű tartalom esetén a `hu` értéket kell beírni, angol nyelvű tartalom esetén az `en` értéket.

A HTML-tagen belül két fő egységre tagolódik a tartalom, a **fejre** (`<head>`) és a **törzsre** (`<body>`).

A fej részben (`<head>`) kell beállítani például a karakterkódolást. Itt érdemes az UTF-8 kódolást használni, hogy a magyar ékezetes karakterek megfelelően jelenjenek meg az oldalon. Ehhez viszont a szerkesztőprogramban is ugyanezt a karakterkódolást kell beállítanunk a mentés előtt.

A `<title>` tagben az oldal címét kell megadni, amely megfelelően utal a weblap tartalmára.

A `<body>` és `</body>` tagek közti rész a dokumentumtörzs. Ebben helyezhetjük el a lap tartalmát, a címsorokat, bekezdéseket, képeket, videókat stb.

**Fontos!** A `<body>` tagben elhelyezett szöveget normál esetben nem tudjuk formázni sortörésekkel, tabulátorokkal, szóközökkel, vagyis ezek hatása nem fog megjelenni a böngészőprogramban. Térközöket a megfelelő tagek használatával, illetve majd a stíluslapok használatával tudunk beállítani.

## Készítsünk közösen egy weblapot!

A folytatásban készítsünk egy statikus honlapot, amelynek kapcsán megismerjük a legfontosabb címkéket!

A honlap a golden retriever kutyafajtáról szól. A hozzávalóit a letölthető állományok között találod a *goldi* mappában. Ebben megtalálhatók a felhasználható szövegek, a képek (képek mappa) és a videók (videók mappa) is.

Az *index.html* állományban megtaláljuk a korábban bemutatott alapstruktúrát. Nyissuk meg ezt az állományt a kódszerkesztő alkalmazásban!

Módosítsuk az oldal címét (a `<title>` tag tartalmát) úgy, hogy a szöveg utaljon a honlap tartalmára. Pl. így:

```
<title>Golden Retrieverek - Kezdőlap</title>
```

Először a honlap tartalmi részét fogjuk elkészíteni, és csak utána térünk rá arra, hogyan nézzen ki az oldal. Ezért kezdetben még nem a képen látható módon jelenik meg az oldal. Látni fogjuk ugyan a címsorokat, bekezdéseket, a fotókat és a videót is, csak az alapértelmezett megjelenéssel, fehér háttérrel, fekete szöveggel.

### Címsorok használata (`<h1>`, `<h2>`...)

A szövegszerkesztés témakörben megtanultuk, hogy a dokumentumot a címsorok segítségével kell tagolnunk annak érdekében, hogy az olvasó könnyen eligazodjon a tartalomban. Ez a weboldalak esetén sincs másként.

Kezdjük azzal, hogy elhelyezünk egy egyes szintű címsort, mindjárt a `<body>` nyitótag után. Erre a `<h1>` tag szolgál.

Szintén adjuk meg most az oldal alcímsorait is! Ezek kerüljenek kettes címsorba a `<h2>` taggal! Összesen hat címsorszintet lehetne használni a HTML nyelvben, vagyis akár a `<h6>` címkét is használhatnánk.

```
<h1>A golden retriever bemutatása</h1>
```

```
<h2>Előnyös tulajdonságai</h2>
```

```
<h2>Goldi, a kutya</h2>
```

► A begépelendő kód

Mentsük el az állományt, és nyissuk meg a böngészőprogramban! Ezt megtehetjük úgy, hogy a fájlt ráhúzzuk a böngészőprogram ablakára, de azt is megfelelő, ha duplán rákattintunk az állományra. Utóbbi esetben az alapértelmezett böngészőprogramban jelenik meg az oldal. Ha mindent jól csináltunk, akkor megjelennek a címsorok a böngészőprogramban. A címsorokat a böngésző nagyobb, félkövér betűkkel jeleníti meg.



► Az elkészítendő honlap kezdőlapjának képe



## Megjegyzés beírása a forráskódba (<!-- -->)

Ha szeretnénk megjegyzést tenni a forráskódba, akkor ezt az `<!--` és `-->` jelek közé kell tennünk. A megjegyzésbe írt szöveg nem jelenik meg a böngészőben. Ha átmenetileg ki szeretnénk venni a megjelenítésből egy kódrészletet, akkor a megjegyzés jelekkel körbekeríthetjük. Azt azonban fontos tudnunk, hogy ha valaki megnézi a böngészőprogramban a forráskódot, akkor megtalálja, hogy milyen szöveget, illetve kódot tettünk megjegyzésbe. A megjegyzésbe tett szöveget általában eltérő színnel (pl. zöld) jelzik a kódszerkesztő programok.

A `<body>` tag után tegyünk be egy megjegyzést a kódba, amelyben a saját nevünk szerepeljen!

```
<!-- Nagy Tamara -->
```



## Bekezdés és sortörés használata (<p>, <br>)

Helyezzük el az első bekezdést a címsor alá! Használjuk ehhez a `<p>` taget! Próbáljuk ki a `<br>` tag hatását a bekezdésen belül, amely egy sortörést hoz létre. Ezt a „származik.” szó után helyezzük el. Vigyázzunk, ennek a címkének nincsen záró párja!

Ellenőrizzük a címke hatását a böngészőprogramban úgy, hogy frissítjük az oldalt!

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

```
<p>A golden retriever egy kedves, barátságos kutya fajta, mely Skóciából származik. <br>A fajta tudományos megnevezése: golden retriever - Canis lupus familiaris.</p>
```

## Egyszerű szövegformázások (<b>, <i>)

Ismerkedjünk meg két új címkével, amelyekkel alapvető szövegformázásokat végezhetünk el. A `<b>` tag a szöveg félkövér formázására alkalmas. Főként kulcsszavak kiemelésére használatos. Például egy bekezdésen belül azon szavakat érdemes így kiemelni, amelyek fontosak a megértés szempontjából. A honlap látogatója ezen szavakat végigpásztázva gyorsabban áttekintheti a szöveget.

Az `<i>` tag segítségével dőlt betűs formázást állíthatunk be. Akkor használjuk, ha egy szövegrész hangulatban, hangnemben eltér a szöveg többi részétől. Az élőlények latin nevének leírásakor is ezt a címkét kell használni.

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

```
<p>A <b>golden retriever</b> egy kedves, barátságos kutya fajta, mely Skóciából származik. <br>A fajta tudományos megnevezése: golden retriever - <i>Canis lupus familiaris.</i></p>
```

Félkövér és dőlt formázás azonban nem csak a fenti címkék használatával valósítható meg. De vigyázzunk, a hasonló megjelenés mögött, más-más jelentés lehet, amely a böngészőprogramokban nem vehető észre. Ha viszont egy vak felhasználó képernyőolvasó programot használ, akkor máshogy olvashatja fel a program ezeket a szövegeket.



## Fontos, hangsúlyos szövegek jelölése (<strong>, <em>)

A `<strong>` címkével olyan szövegeket emelhetünk ki, amelyen nagyon fontosak, ezért erősen ki szeretnénk emelni. Ezek megjelenítése éppúgy félkövér formázással történik a böngészőben, mint a `<b>` tag esetén.

Az `<em>` címke a hangsúlyos kiemelésre szolgál. Az élő beszédben is sokszor nyomtatékosítunk egy-egy szót, amely a mondat jelentését is befolyásolhatja.

Például az alábbi mondatoknak kissé módosul a jelentése, ha más-más szavakat hangsúlyozunk. Tegnap Ábel *hiányzott* az iskolából. Tegnap *Ábel* hiányzott az iskolából.

Az ilyenfajta hangsúlyok jelzésére alkalmas tehát az `<em>` tag.

*A példánkban szereplő bekezdésben jelöljük meg, hogy melyek a fontos, hangsúlyos elemek!*

```
<p><strong>Fontos!</strong> Ha gazdi szeretnél lenni, nézz utána,
hogya <em>felelős</em> kutyatartással kapcsolatban milyen ajánlások
és kötelezettségek vannak!</p>
```

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

## Felsoroláslista használata (<ul>, <li>)

A következő téma a kutyusok előnyös tulajdonságairól szól. Ezeket egy felsoroláslistában fogjuk elhelyezni.

Felsoroláslistát úgy készíthetünk, hogy egy `<ul>` címkét nyitunk meg, és abban helyezük el a listaelemeket, amelyek `<li>` tagekbe kerülnek. Az `ul` az unordered list (felsorolás lista), a `li` a list item (listaelem) szavak rövidítése.

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

```
<ul>
  <li>Okos, könnyen képezhető</li>
  <li>Szelíd, családbarát</li>
  <li>Nagyon jó vadászkutya</li>
</ul>
```

## Sorszámozott lista használata (<ol>, <li>)

A sorszámozott listák nagyon hasonlóan hozhatóak létre, mint a felsoroláslisták, csak ebben az esetben az `<ol>` taget kell használni. Az `ol` az ordered list (sorszámozott lista) szavak rövidítése.

A sorszámozott lista általános megadási módja:

```
<ol>
  <li>listaelem</li>
  <li>listaelem</li>
  <li>listaelem</li>
</ol>
```

*A példánkban most nincs ilyen lista, ezért folytassuk a képekkel!*

## Képek beillesztése (<img>)

A honlapokon képeket is elhelyezhetünk az `<img>` tag használatával. A legfontosabb paraméterei:

- src** A kép elérési útvonalát tartalmazza.
- alt** Alternatív szöveg, vagyis a kép rövid tartalmi leírását tartalmazza.
- title** A képhez rendelt cím, amely a böngészőkben akkor jelenik meg, ha a kép fölé visszük az egeret.
- width** A kép szélessége képpontokban.
- height** A kép magassága képpontokban.

### A kép forrása (src)

A kép forrását az **src** paraméterben kell megadnunk. Ennek egyik módja az **abszolút útvonalmegadás** (a kép teljes webcímének beírásával), de ennek előfeltétele, hogy a kép már publikálva legyen a világhálón. Pl. <http://tiny.cc/8mqblz>

Ha a honlapunk alkönyvtárszerkezete alapján adjuk meg az elérési utat, akkor **relatív útvonalmegadást** alkalmazunk. Fontos, hogy annak a HTML-állománynak az elhelyezkedéséhez képest adjuk meg a kép elérési útját, amelyben hivatkozni szeretnénk rá. Lássunk néhány példát!

- src="csaladikep.jpg"** Ebben az esetben a kép ugyanabban a mappában van, mint a HTML-állomány.
- src="kepek/csaladikep.jpg"** A kép most a kepek nevű mappában van. Ez a mappa pedig abban a könyvtárban van, amelyben a HTML-állomány.
- src="../kepek/csaladikep.jpg"** A kép most a kepek nevű mappában van. Ez a mappa pedig egy szinttel feljebbi könyvtárból nyílik, ahhoz képest, ahol a HTML-állomány található.

**Fontos!** Az a könyvtár, amelyben az `index.html` állományt elhelyezzük, lesz a gyökérkönyvtára a honlapunknak. Ezen belül tetszőleges könyvtárstruktúrát kialakíthatunk, de ne hivatkozzunk az `index.html` állományban (és a vele egy szinten lévő HTML-állományokban) olyan könyvtárra, amely feljebbi szinteken lévő alkönyvtárakra mutat (pl. `../kepek`), mert ez a weblap publikálásánál problémákat okozhat!

*Illesszünk be most két képet egymás mellé, azonos magasságban. A magasság legyen 200 képpont! Adjuk meg az alt és title paramétereket is!*

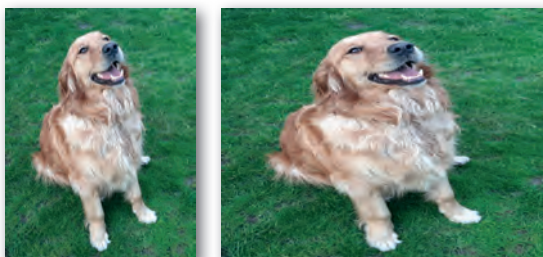
```


```



Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

**Fontos!** Amennyiben a szélesség (**width**) és a magasság (**height**) közül csak az egyiket adjuk meg, akkor a böngésző aránytartóan átméretezi a képet. Ha mindkét paramétert megadjuk, akkor vigyázzunk arra, hogy a kép arányai megváltozhatnak, ami a kép torzulásához vezethet.



► Kép átméretezése aránytartó és torzított formában

### Ábrák, illusztrációk jelölése feliratokkal (<figure>, <figcaption>)

A HTML5-szabványban a <figure> címkét használhatjuk arra a célra, hogy illusztrációt adjunk meg. Ezen illusztrációhoz egy feliratot is társíthatunk a <figcaption> címkével, amelyet a <figure> tagen belül kell elhelyeznünk.

Ha az illusztrációnk egy (vagy több) kép, akkor természetesen az <img> taget (vagy tageket) kell elhelyeznünk a <figure> tagen belül.

A korábban már beillesztett képeket helyezzük el a <figure> tagen belül, és adjuk meg a kép feliratát is. Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!



```
<figure>
  <img ...>
  <img ...>
  <figcaption>Fotók Goldiról</figcaption>
</figure>
```

**Fontos!** Az illusztrációk nemcsak fotók lehetnek. Grafikonokat, versrészleteket, programkód részleteket és más elemeket is elhelyezhetünk ebben a címkében.

### Hivatkozások (linkek) megadása (<a>)

A hivatkozások (linkek) segítségével el tudjuk érni, hogy az oldalunkról egy másik erőforrásra (weboldalra, médiaelemhez, egyéb fájlhoz) lehessen eljutni. Oldalon belüli ugrásra is használhatunk linkeket.

**Érdekesség:** Az angol link szó láncot jelent magyarul. A lánc pedig sokszor egy horgonyhoz kapcsolódik (pl. a hajók esetén). A horgonyt angolul anchornak nevezik. Ennek a rövidítéséből jött létre az <a> tag.



A link létrehozása a következő módon történik: `<a href="url">Link szövege</a>`  
Vagyis a `href` paraméterben kell megadnunk a hivatkozott erőforrás URL-jét. A címke nyitó és záró párja közti szöveg lesz a link szövege, amelyre majd rá tudunk kattintani.

*Készítsünk egy olyan linket, amelyet követve a honlapunk látogatói még több kutyás képet nézhetnek meg. A weboldal, amelyre hivatkozunk, a <http://tiny.cc/goldipix> rövidített címen elérhető Pixabay portál. A <https://pixabay.com/> portálon jogtiszta, ingyenesen felhasználható fotók találhatók. Csak rá kell keresned egy témára, és máris rengeteg jó minőségű fotót kapsz eredményül.*

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

```
<p>Ha szeretnél még több kutyás fotót nézegetni, látogasd meg a  
<a href="http://tiny.cc/goldipix">Pixabay portál - Goldenek</a>  
fotó oldalát!</p>
```

### Hivatkozás oldalon belül

Azt is megtehetjük, hogy a hivatkozás segítségével **a lap egy adott pontjára ugrunk**. Ehhez azt a helyet el kell neveznünk az `id` paraméterrel. Ennek a névnek egyedinek kell lennie a teljes oldalon belül. Ezt a paramétert bármelyik címkénél felhasználhatjuk. Vigyázzunk, a név kis- és nagybetű-érzékeny.

Az egyedi nevet a link `href` paraméterében úgy kell megadnunk, hogy egy kettős keresztet teszünk elé (`href="#egyedinév"`).

Lássunk egy példát!

Van egy egyes címsorunk, amelynek a `bevezeto` nevet adtuk.

```
<h1 id="bevezeto">Bevezető</h1>
```

Ha erre a pontra szeretnénk ugrani, akkor az `<a>` tag `href` paraméterében ugyanezt a nevet kell megadnunk, de elé egy kettős keresztet (hashtaget) kell írunk.

```
<a href="#bevezeto">Ugrás a bevezetőre</a>
```

Ha egy másik oldal (pl. `magamrol.html`) egy pontjára szeretnénk ugrani, akkor az oldal elérhetősége után kell megadnunk a `#` jelet és a nevet.

```
<a href="magamrol.html#bevezeto">Ugrás a bevezetőre</a>
```

*Helyezzünk el a weblapunk alján egy olyan linket, amellyel vissza tudunk ugrani az oldal elejére! Ehhez az egyes címsornak adjunk egy egyedi azonosítót, és a hivatkozásban ugyanezt a nevet adjuk meg!*

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

```
<h1 id="eleje">A golden retriever bemutatása</h1>  
<p><a href="#eleje">Vissza az oldal tetejére</a></p>
```

## Hivatkozás saját oldalra, több oldalból álló weboldalak

Fejlesszük tovább úgy a weboldalt, hogy ne csak egy oldalból álljon. Mentsük el a korábban látott alapstruktúrát `tipusok.html` néven ugyanabba a mappába, mint ahol az `index.html` állomány van. Majd módosítsuk az `index.html` állományt, hogy a másik lapra is el lehessen jutni. Soroljuk be ezt a bekezdést a „menu” osztályba, hogy majd a stíluslap segítségével megváltoztathassuk a kinézetét!

```
<p class="menu"><a href="index.html">Kezdőlap</a> <a href="tipusok.html"> Típusok</a></p>
```

## Videó beillesztése (<video>, <source>)

A HTML5-szabvány jelentős újdonsága, hogy a multimédiás állományok beillesztését (pl. hangok, videók) jelentősen leegyszerűsíti.

Videókat a `<video>` taggel tudunk az oldalba illeszteni. Hogy egy vezérlő eszköztár is megjelenjen (lejátszás, megállítás, hangerő stb. gombokkal), a `controls` paramétert is használnunk kell. Ez egy logikai paraméter, így elég leírni a nevet, nem kell értéket adnunk.

A `<video>` tagen belül a `<source>` tag használatával adhatjuk meg a videó forrását. Ezen tag `src` paraméterében kell megadnunk a videó elérési útját. A legnagyobb böngészőtámogatottsága az mp4-formátumú videóknak van, így célszerű ilyen formátumban létrehozunk a videót. Az mp4 formátumú videók esetén a `video/mp4` értéket kell megadnunk a `type` (típus) paraméterben.

A `<video>` esetén is megadható a szélesség (`width`) és magasság (`height`) paraméter csakúgy, mint a képek esetén.

A videó beillesztésének általános módja tehát:

```
<video controls width="" height="">
  <source src="" type="video/mp4">
</video>
```

Próbáljuk ki a videóbeillesztést! A médiaelemek között találunk egy videót Goldiról. Hozunk létre egy bekezdést, amelyben bevezetjük szövegesen a videó tartalmát, majd ágyazzuk be az oldalra a videót!

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban!

```
<p>A videón is láthatod, hogy Goldi nagyon szeret a földön hempe-  
regni, amely miatt általában koszosan jövünk haza a sétából.</p>
<video controls width="400">
<source src="videok/goldiseta.mp4"
  type="video/mp4">
</video>
```



## Hangok beillesztése (<audio>, <source>)

Hangokat nagyon hasonló módon lehet beilleszteni, mint a videókat. Ebben az esetben az <audio> taget kell használni, és érdemes mp3-formátumú állományokat használni a széles körű böngészőtámogatottság miatt.

A hang beillesztésének általános módja:

```
<audio controls>  
  <source src="" type="audio/mpeg">  
</audio>
```

**Figyelem!** Ha videókat, illetve hangokat illesztés be az oldalra, akkor abban az esetben lesz akadálymentes a weboldal, ha a videó vagy hangállomány tartalmát szövegesen is leírod. Ha a videó rövid, akkor magában a weboldal szövegében is lehet utalni arra, hogy mi látható a videón. Hosszú leírást érdemes külön weblapon elkészíteni, és az adott videó vagy hang alá elhelyezni egy hivatkozást, amely erre a leírásra mutat. Videó esetén egy forgatókönyvszerű leírást érdemes mellékelni, hang esetén pedig az elhangzó szöveget érdemes leírni.



## Táblázatok használata (<table>, <tr>, <th>, <td>)

Ha táblázatos formában szeretnénk elhelyezni adatokat a weboldalunkon, akkor több címkével is meg kell ismerkednünk.

A <table> tagben kell elhelyeznünk a teljes táblázatot. Ebben megadhatjuk a táblázat feliratát (<caption>) is.

A táblázat sorait a <tr> címke jelöli. A táblázat soraiban elhelyezhetünk fejléccellákat (<th>), amelyek leírják, hogy az adott sorban, illetve oszlopban milyen jellegű adatok vannak. Az adatokat tartalmazó cellákat pedig <td> tagek közé kell zárunk. Az egy sorban lévő elemeket balról jobbra haladva kell leírunk.

Nézzünk egy példát! Ha az alábbi táblázatot szeretnénk egy honlapon elhelyezni, amely az iskolai futóverseny eredményeit tartalmazza, akkor az itt látható kódot kell használnunk.



Az 5 km-es futóverseny eredményei

Helyezés	Név	Osztály
1.	Kiss Réka	9.C
2.	Hegedűs Ábel	9.A
3.	Horváth Kristóf	9.B

```

<table>
<caption>Az 5 km-es futóverseny eredményei</caption>
<tr>
    <th>Helyezés</th>
    <th>Név</th>
    <th>Osztály</th>
</tr>
<tr>
    <td>1.</td>
    <td>Kiss Réka</td>
    <td>9.C</td>
</tr>
<tr>
    <td>2.</td>
    <td>Hegedűs Ábel</td>
    <td>9.A</td>
</tr>
<tr>
    <td>3.</td>
    <td>Horváth Kristóf</td>
    <td>9.B</td>
</tr>
</table>

```

## Feladatok

Készítsük el önállóan az itt látható táblázatot, és helyezzük el a `tipusok.html` oldalon! Ne csak a táblázatot valósítsuk meg, hanem a címsort, a főmenüt és a forrásra mutató hivatkozást is!

A Golden Retriever bemutatása	
<u><a href="#">Kezdőlap</a></u> <b>Típusok</b>	
<b>Típusok</b>	
A Golden Retriever típusai	
Típus	Típus leírása
Munkatípus	Könnyedebb felépítésű. Vékonyabb, hosszabb fej, sötétebb szőrszín, élénk temperamentum, karcsúbb, ruganyosabb, gyorsabb.
Kiállítási típus	Dúsabb, hosszabb szőrzet, robusztusabb felépítés, világosabb szőrszín, határozottabb.
Európai típus	Szőrszíne az arany világosabb árnyalata. Rövidebb szőrű, mint az amerikai. Lába rövidebb az amerikalnál, így megjelenése zömökebb. Jó pigmentált. Jó szögellésű elől, hátul.
Amerikai típus	Orr és fej része finomabb. Színe közepes, rézvörös, de sosem krém. Mellso szögellésel szerényebbek, hátul olykor túlszögelt.
Forrás: <a href="http://hu.wikipedia.org/wiki/Golden_retriever">hu.wikipedia.org/wiki/Golden_retriever</a> <a href="#">Vissza az oldal tetejére</a>	

## A legfontosabb HTML5-címkék összefoglaló táblázata

Tag (címké)	Leírás
<code>&lt;p&gt;Bekezdés&lt;/p&gt;</code>	Bekezdés ( <i>paragraph</i> )
<code>&lt;h1&gt;Címsor 1&lt;/h1&gt;</code>	Egyes címsorszint ( <i>heading</i> )
<code>&lt;h6&gt;Címsor 6&lt;/h6&gt;</code>	Hatos címsorszint ( <i>heading</i> ) (a közbenső szintek értelemszerűen <code>h2</code> , <code>h3</code> , <code>h4</code> , <code>h5</code> )
<code>&lt;b&gt;szöveg&lt;/b&gt;</code>	Félkövér ( <i>bold</i> ) szöveg (kulcsszavak jelölése)
<code>&lt;i&gt;szöveg&lt;/i&gt;</code>	Dőlt ( <i>italic</i> ) kiemelés (eltérő hangulatú szöveg, latin nevek)
<code>&lt;strong&gt;fontos szöveg&lt;/strong&gt;</code>	Fontos, erősen kiemelt ( <i>strong</i> ) szöveg
<code>&lt;em&gt;hangsúlyos szöveg&lt;/em&gt;</code>	Hangsúlyosan kiemelt ( <i>emphasis</i> ) szöveg
<code>&lt;br&gt;</code>	Sortörés ( <i>break</i> )
<code>&lt;img src="uri" alt="leírás" width="" height=""&gt;</code>	A megadott forráson ( <i>source</i> ) elérhető kép ( <i>image</i> ) beszúrása leírással ( <i>alternate</i> ), megadott szélességben ( <i>width</i> ), magasságban ( <i>height</i> ).
<code>&lt;figure&gt;   (ide illesztjük be a képet, stb.)   &lt;figcaption&gt;Felirat &lt;/figcaption&gt; &lt;/figure&gt;</code>	Illusztráció ( <i>figure</i> ), felirattal ( <i>figure caption</i> )
<code>&lt;ul&gt;   &lt;li&gt;listaelem&lt;/li&gt; &lt;/ul&gt;</code>	Felsoroláslista ( <i>unordered list</i> ), listaelemmel ( <i>list item</i> )
<code>&lt;ol&gt;   &lt;li&gt;listaelem&lt;/li&gt; &lt;/ol&gt;</code>	Sorszámozott lista ( <i>ordered list</i> ), listaelemmel ( <i>list item</i> )
<code>&lt;a href="uri"&gt;Link szövege&lt;/a&gt;</code>	Hiperhivatkozás (link) ( <i>a=anchor</i> , <i>href=hyperlink reference</i> )
<code>&lt;video controls width="" height=""&gt;   &lt;source src="mp4_videó_url" type="video/mp4"&gt; &lt;/video&gt;</code>	Videó (mp4) ( <i>video</i> ) beillesztése, vezérlő eszköztárral ( <i>controls</i> ), megadott szélességben ( <i>width</i> ), magasságban ( <i>height</i> ). A forrás elérhetőségét ( <i>source</i> ) és típusát ( <i>type</i> ) is meg kell adjuk.
<code>&lt;audio controls&gt;   &lt;source src="mp3_hang_url" type="audio/mpeg"&gt; &lt;/audio&gt;</code>	Hangállomány (mp3) ( <i>audio</i> ) beillesztése, vezérlő eszköztárral ( <i>controls</i> ). A forrás elérhetőségét ( <i>source</i> ) és típusát ( <i>type</i> ) is meg kell adjuk.
<code>&lt;!-- megjegyzés --&gt;</code>	Megjegyzés ( <i>comment</i> ) elhelyezése a HTML-kódban
<code>&lt;table&gt; &lt;caption&gt;&lt;/caption&gt; &lt;tr&gt;   &lt;th&gt;&lt;/th&gt;   &lt;td&gt;&lt;/td&gt; &lt;/tr&gt; &lt;/table&gt;</code>	Táblázat ( <i>table</i> ) elhelyezése: <ul style="list-style-type: none"> <li>• <code>table</code>: táblázat</li> <li>• <code>caption</code>: felirat</li> <li>• <code>tr</code>: táblázat sora (<i>table row</i>)</li> <li>• <code>th</code>: fejléccella (<i>table heading</i>)</li> <li>• <code>td</code>: adatcella (<i>table data</i>)</li> </ul>

## A stíluslapok (CSS) használata

*Most már elkészült a honlapunk tartalma. De hogy kapja meg azt a kinézetet, amelyet a korábbi képen láthattunk? Csatoljunk hozzá egy megfelelő stíluslapot! A css mappában megtalálható az a stíluslap (goldi.css), amely a fotón látható kinézetet eredményezi. Csatoljuk hozzá a HTML-állományhoz.*

*A stíluslapok csatolása sokféle módon történhet a dokumentumokhoz. Most a külső stíluslap csatolását mutatjuk meg.*

### A <link> tag használata

A külső stíluslapokat kétféle módon is hozzácsatolhatjuk a HTML-állományokhoz. Az első módszer, hogy használjuk a <link> taget a HTML-állomány <head> részében, az alábbi módon:

```
<link rel="stylesheet" type="text/css" href="url">
```

A href paraméterben kell megadnunk a külső stíluslapállomány elérhetőségét.

A mi esetünkben a CSS-állomány a css mappán belül helyezkedik el, goldi.css néven. Ezért a következő kódot kell használnunk az oldal <head> részében:

```
<link rel="stylesheet" type="text/css" href="css/goldi.css">
```

Próbáljuk ki a kódot, és ellenőrizzük az oldal megjelenését a böngészőprogramban! Ugye megváltozott az oldal kinézete a korábban látott képnek megfelelően?

### A @import szabály használata

Külső stíluslapot a CSS-szabvány segítségével is lehet csatolni a dokumentumhoz. Ahhoz, hogy ezt kipróbáljuk, tegyük megjegyzésjelek közé a <link> taget vagy töröljük ki!

Ezt a módszert követve, a <head> részben el kell helyezni egy lapon belüli stílusleírást a <style> taggel. Ennek az elején lehet használni a @import szabályt.

```
<style>
  @import url("");
</style>
```

A mi esetünkben a következő kódot kell használnunk:

```
<style>
@import url("css/goldi.css");
</style>
```

Próbáljuk ki a kódot, és mentjük el az eredményt! Frissítsük az oldalt a böngészőben! Ugye most is érvényre jut a csatolt stíluslap?



## A stíluslap módosítása

A korábbi fejezetben bemutattuk, hogy hogyan tudunk a CSS-szabvány segítségével kijelölni elemeket (szelektorok), és hogyan adhatjuk meg a tulajdonságokat a deklarációs blokkban. Most ismerkedjünk meg a témával a gyakorlatban is!

### Feladatok

Nyissuk meg a `goldi.css` állományt abban a szerkesztőprogramban, amelyet a HTML-állományok szerkesztésére is használtunk. Megnyitás után láthatjuk, hogy milyen CSS-szabályokat alkalmaztunk a stíluslapállományban. A jobb megértés érdekében minden sorhoz egy magyarázatot is elhelyeztünk.

Annak érdekében, hogy stíluslapot tudjunk készíteni, vagy meglévőt módosítani, az alábbi táblázatban elhelyeztük a leggyakrabban használt tulajdonságokat és magyarázatukat.

<code>text-align</code>	A szöveg vízszintes igazításának módja. Értékei: <code>left</code> (balra), <code>center</code> (középre), <code>right</code> (jobbra), <code>justify</code> (sorkizárt módon). Példák: <code>text-align:center;</code> <code>text-align:right;</code>
<code>font-size</code>	A betű mérete. Megadható például képpontokban (px), vagy akár százalékosan (%) is. Használható az <code>em</code> mértékegység is, amellyel az aktuális betűméret valahányszorosát állíthatjuk be. Példák: <code>font-size:16px;</code> <code>font-size:120%;</code> <code>font-size:1.5em;</code>
<code>font-weight</code>	A szöveg félkövér megjelenítése. Tipikus értéke a <code>'bold'</code> . Ha vissza akarjuk állítani a normál megjelenést, használjuk a <code>'normal'</code> értéket! Példák: <code>font-weight:bold;</code> <code>font-weight:normal;</code>
<code>font-style</code>	A szöveg dőlt megjelenítésére szolgál. Tipikus értéke az <code>'italic'</code> . Ha vissza akarjuk állítani a normál megjelenést, használjuk a <code>'normal'</code> értéket! Példák: <code>font-style:italic;</code> <code>font-style:normal;</code>
<code>font-family</code>	A betűtípus beállítására szolgál. Konkrét betűtípusokat fel lehet sorolni, a végén egy általános betűcsaládot kell megadni (pl. <code>serif</code> = talpas betűk, <code>sans-serif</code> = talp nélküli betűk). Ha a betűtípus neve szóközt tartalmaz, akkor aposztrófjelek közé kell tenni. Példák: <code>font-family:Arial,Verdana,sans-serif;</code> <code>font-family:'Times New Roman',Times,serif;</code>
<code>border</code> <code>border-top</code> <code>border-right</code> <code>border-bottom</code> <code>border-left</code>	Az elemet körbevevő szegély beállítására szolgál. Megadhatjuk a szegély vastagságát képpontokban (px), a szegély stílusát ( <code>solid</code> = folytonos, <code>dotted</code> = pontozott, <code>dashed</code> = szaggatott, <code>double</code> = dupla stb.), valamint a színét. Ha a <code>border</code> tulajdonságot használjuk, akkor mind a négy oldali szegély ugyanolyan lesz. Ha nem ezt akarjuk, akkor használhatjuk csak az adott oldalra vonatkozót ( <code>top</code> =felső, <code>right</code> =jobb, <code>bottom</code> =alsó, <code>left</code> =bal) Példák: <code>border-top:1px solid blue;</code> <code>border:4px double #8b0000;</code>
<code>border-radius</code>	A szegély lekerekítettségét állítja be. Például: <code>border-radius:20px;</code>
<code>float</code>	Az adott elemet lehet balra vagy jobbra lebegtetni. A lebegtetés azt jelenti, hogy a környező elemek körbefolyják az adott elemet. Például: <code>float:left;</code> <code>float:right;</code>



padding padding-top padding-right padding-bottom padding-left	Kitöltés megadása, ami a tartalom és az ezt körbevevő szegély közti térközt jelenti. Belső margónak is hívják. Ha a <code>padding</code> tulajdonságot használjuk, és egy értéket adunk meg, akkor mind a négy oldali beállítás ugyanakkora lesz. Ha nem ezt akarjuk, akkor használhatjuk csak az adott oldalra vonatkozót.
margin margin-top margin-right margin-bottom margin-left	A margó megadására szolgál. A margó az adott elem szegélye és az őt körülvevő többi elem közti távolságot jelenti. Ha a <code>margin</code> tulajdonságot használjuk és egy értéket adunk meg, akkor mind a négy oldali beállítás ugyanakkora lesz. Ha nem ezt akarjuk, akkor használhatjuk csak az adott oldalra vonatkozót. Az elemek középre igazítására is a margóbeállítást kell használni, ilyenkor <code>'auto'</code> értéket kell adni a bal és jobb margónak. Ekkor a <code>margin-left:auto;</code> és <code>margin-right:auto;</code> tulajdonságokat kell beállítani.
color	A szöveg színének beállítására szolgál. Megadhatunk színnevet (pl. blue), de akár RGB színkódot is. A képszerkesztő programokban gyakran a színkódot hexadecimális kóddal is megtaláljuk, amelynek az elején egy # karakter áll. Az alábbi példában ugyanazon szín megadását láthatjuk különböző módokon: <code>color:darkred;</code> <code>color: #8b0000;</code> <code>color: rgb(139,0,0);</code>
background-color	Háttérszín beállítása. A színek hasonlóan adhatóak meg, mint a szövegszín esetén.
width	Az elem szélességét állítja be. A méret megadható képpontokban és akár százalékban is. Példák: <code>width:600px;</code> <code>width:90%;</code>
height	Az elem magasságát állítja be. Hasonlóan állítható be, mint a szélesség.
min-width min-height	Az elem minimális szélességét, illetve magasságát jelenti, amely alá nem lehet átméretezni a böngészőprogramban.
max-width max-height	Az elem maximális szélességét, illetve magasságát jelenti, amely fölé nem méretezi át a böngésző az elemet.
background-image	A háttérkép beállítására szolgál. A háttérkép elérhetőségét az <code>url()</code> szövegen belül kell megadni, az aposztrófok között. Például: <code>background-image:url('kepek/mancs.png');</code>
background-repeat	A háttérkép ismétlődésének beállítására alkalmas. Az értékei a következők lehetnek: <code>repeat</code> (mozaikszerűen ismétlődő), <code>no-repeat</code> (nem ismétlődő), <code>repeat-x</code> (csak vízszintes irányba ismétlődő), <code>repeat-y</code> (csak függőleges irányba ismétlődő). Például: <code>background-repeat:no-repeat;</code>
background-position	háttérkép bal felső sarkának koordinátái. Először az x tengelyre vonatkozó, utána az y tengelyre vonatkozó értéket kell beállítani. Például: <code>background-position:0 20px;</code>

## Feladatok

Készítsünk másolatot a `goldi.css` állományból, és csatoljuk ezt az új állományt a már elkészült HTML-oldalakhoz. Módosítsuk az oldal kinézetét saját elképzeléseinknek megfelelően. Próbáljunk ki új színsémákat, igazítási módokat!

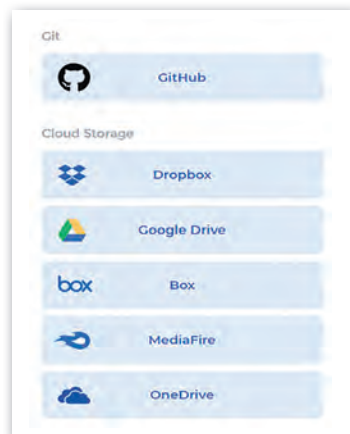
## A statikus honlap publikálása

Ahhoz, hogy az általunk készített statikus honlap mindenki számára elérhető legyen a világhálón, publikálni kell azt egy webes kiszolgálón (webszerveren). Ez azt jelenti, hogy egy segédprogram segítségével csatlakozni kell a webszerverhez, és a megfelelő mappájába fel kell tölteni az összes állományt.

Vannak olyan ingyenes szolgáltatások (pl. fast.io), amelyek lehetővé teszik a statikus honlapok publikálását egy webcímen.

Ilyenkor a regisztráció után meg kell adnunk, hogy milyen webcímen szeretnénk elérhetővé tenni a honlapunkat, majd az ahhoz tartozó összes állományt a portálon ismertetett módon fel kell töltenünk.

Ne feledjük, a kezdőlapot `index.html` néven kell elneveznünk!



- ▶ A fast.io szolgáltató sokféle tárhelyhez tud doménnevet rendelni

### Feladatok, kérdések

1. Derítsük ki, hogy a saját iskolánkban van-e lehetőség a honlapok publikálására. Ha nincs, akkor publikáljuk az elkészített honlapot egy ingyenesen elérhető szolgáltató segítségével (pl. fast.io, gitHub.com)!
2. Három-négy fős csoportokban készítsünk el tetszőleges témában egy honlapot az alábbi elvárások szerint:
  - A weblap legalább két oldalból álljon! Mindegyik oldalba legyen beillesztve egy menü, amellyel el lehet jutni a többi oldalra.
  - Az oldalcímeket precízen töltsük ki mindegyik oldal esetén!
  - A tartalom legalább öt bekezdésből álljon! A bekezdésben lévő szöveget formázzuk meg a tanult címkék segítségével!
  - Legyen az oldalon felsorolási lista és sorszámozott lista is!
  - A tartalom címsorokkal legyen tagolva! Használjunk több címszintet!
  - Legyen egy oldalmenü az oldalon, amellyel az oldalon belüli címsorokra rá lehet ugrani!
  - Legyen az oldalon beillesztve kép és videó! A beillesztésnél figyeljünk az akadálymentességi irányelvekre! (Szorgalmi feladatként hangot is elhelyezhetünk az oldalon, ha van a témához kapcsolódó hangállomány.)
  - Legyen egy olyan táblázat beillesztve, amely legalább három sorból és legalább két oszlopból áll! A táblázatnak legyen felirata! Használjunk fejléc- és adatcellákat is!
  - Csatoljunk a honlaphoz egy stíluslapot!
  - Dolgozzunk jogtiszta forrásból! Felhasználhatunk saját készítésű képeket, videókat, hangállományokat is.
  - Minden felhasznált forrást tüntessünk fel és linkeljünk be!
  - Publikáljuk a weblapot, és a webcímet osszuk meg a többi csoporttal is!

A munka felosztásánál ügyeljünk arra, hogy minden tagnak jusson olyan feladat, amelyben tartalmat kell összegyűjteni, és azt a HTML nyelven le kell írni.