DIGITÁLIS KULTÚRA

12. PROGRAMOZÁS ALAPJAI PYTHON NYELVEN

Összeállította: Kolman Krisztián

TARTALOMJEGYZÉK:

A PYTHON MINT PROGRAMOZÓ NYELV

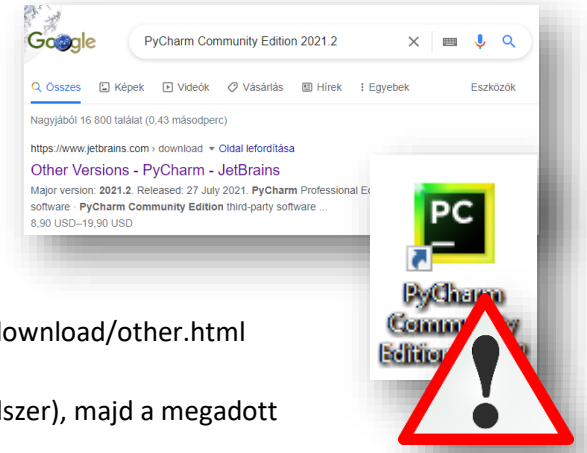
A Python egy magas szintű programozási nyelv, melyet egy holland programozó kezdett el fejleszteni. Könnyen tanulható, hatékony programozási nyelv. A Python egyszerű objektum-orientált programozási megközelítést és nagyon hatékony magas szintű adatszerkezeteket használ. A Python programozásra jellemző, hogy tömör szintaxist és dinamikus gépelést is használ.

Mi az előnye a Python programozási nyelvnek a többi hasonló fejlesztő nyelvhez képest?

- Egyszerű és könnyen megtanulható
- Hordozható és bővíthető
- Objektumorientált
- Tesztelési keretrendszer
- Szkriptek és automatizálás
- Nagy adatkezelés

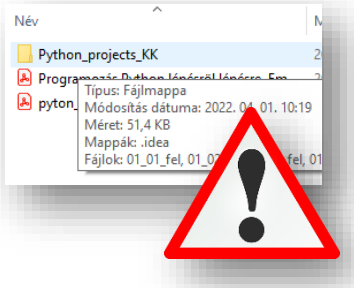
A PYCHARM PROGRAM TELEPÍTÉSE

Ahhoz, hogy Python programozás alapjaival meg tudjunk ismerkedni, meg kell találni a megfelelő keretprogramot. Azt a programot, mellyel elkészítjük a megadott instrukciók alapján, (kódolással, parancsokkal, utasításokkal) a programunkat.

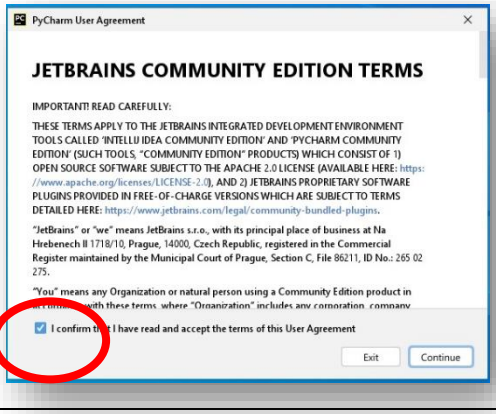


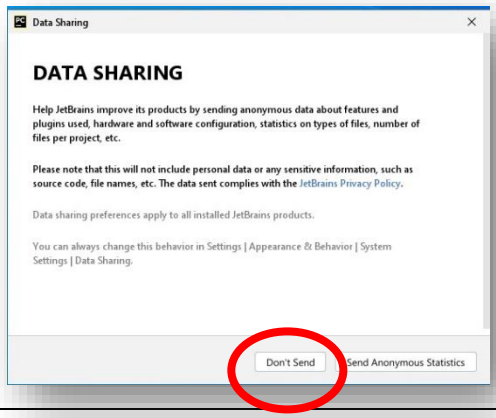
- A PyCharm egy rendkívül népszerű fejlesztői környezet a Python világában. Ezért ezt választottuk.
- Ehhez a PyCharm Community Edition 2021.2 verzióját kellene letölteni. A programból elérhető fizetős (Professional) és ingyenes (Community) verzió is.
- Az első lépés ennek a programnak a telepítése.
  Írjuk a Google keresőbe a program nevét és verzióját!
- Az első találat a: https://www.jetbrains.com/pycharm/download/other.html oldalra vezet.
- Válasszuk ki a megfelelő verziót (szám és operációs rendszer), majd a megadott utasítások alapján telepítsük a programot!

A PYCHARM ELSŐ INDÍTÁSA, A KEZELŐFELÜLET BEÁLLÍTÁSAI

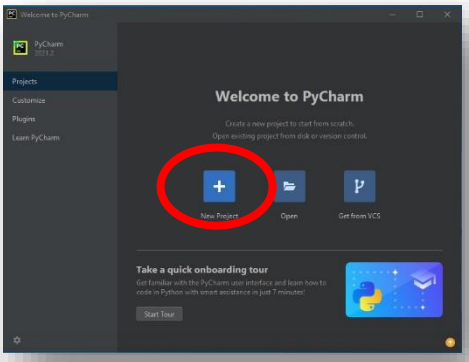


- Mielőtt elindítanánk a programot, hozzunk létre egy mappát! Az elkészített programokat (projekteket) célszerű egy konkrét mappában tárolni, hogy minden egy helyen legyen. Ezért keressünk egy olyan helyet a háttértárolónkon (bárhol), ami megfelel számunkra. Hozzunk létre egy mappát (pl.: Python_projects_XY néven)!
- Indítsuk el az asztalon lévő PyCharm ikont!
  Először meg kell erősíteni és el kell fogadnia a felhasználói szabályokat!
- Aztán a „Don't Send" gombot megnyomni, hogy ne küldjön adatokat a program!



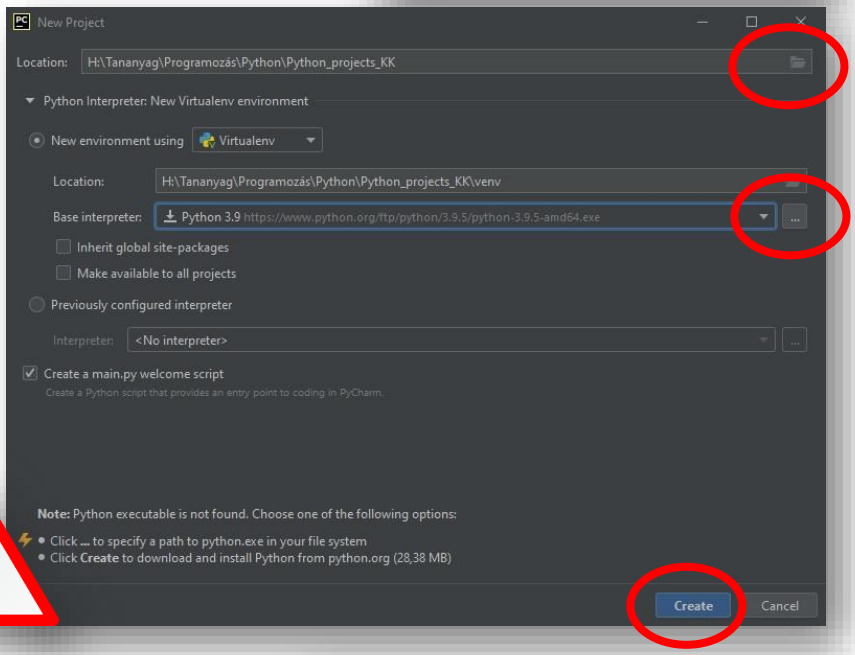


- Az első indításnál egy „Új project”-et hozunk létre!



- A „New Project” ablakban először tallózzunk rá arra a mappára, amit a lépések elején létrehoztunk! Ide fogjuk a fájljainkat elmenteni, és onnan fogjuk megnyitni a meglévő programjainkat!

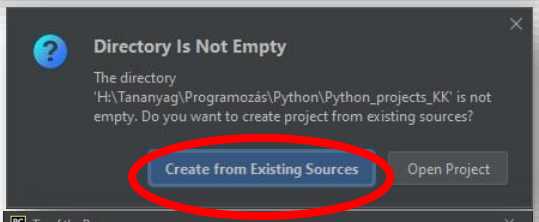
- Az interpretert is ki kell választani, vagy a felajánlottat elfogadni! (A Python úgynevezett interpreteres nyelv, ami azt jelenti, hogy nincs különválasztva a forrás- és tárgykód, a megírt program máris futtatható, ha rendelkezünk a Python értelmezővel.)

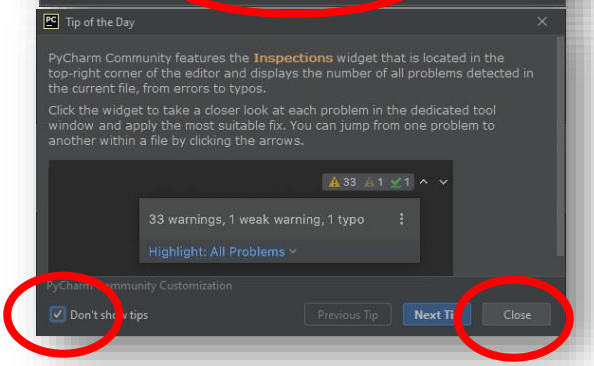


A PyCharm programban készített program fájlok kiterjesztése: *.py

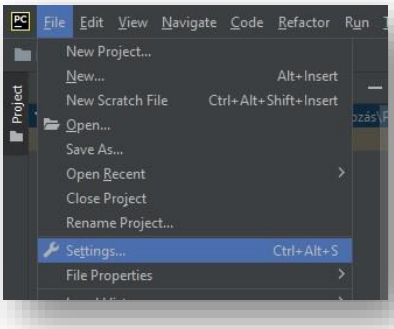
Az ikonjuk:

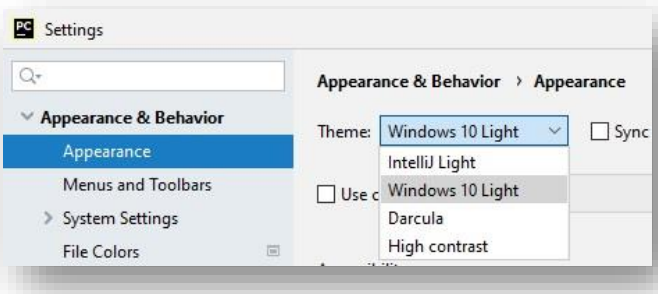
- A „Create” gomb megnyomása után még két gombot meg kell nyomnunk mielőtt nekiállhatnánk programozni! Az egyik, ha a mappa esetleg nem üres. A másik pedig a tippek adásának kikapcsolása!



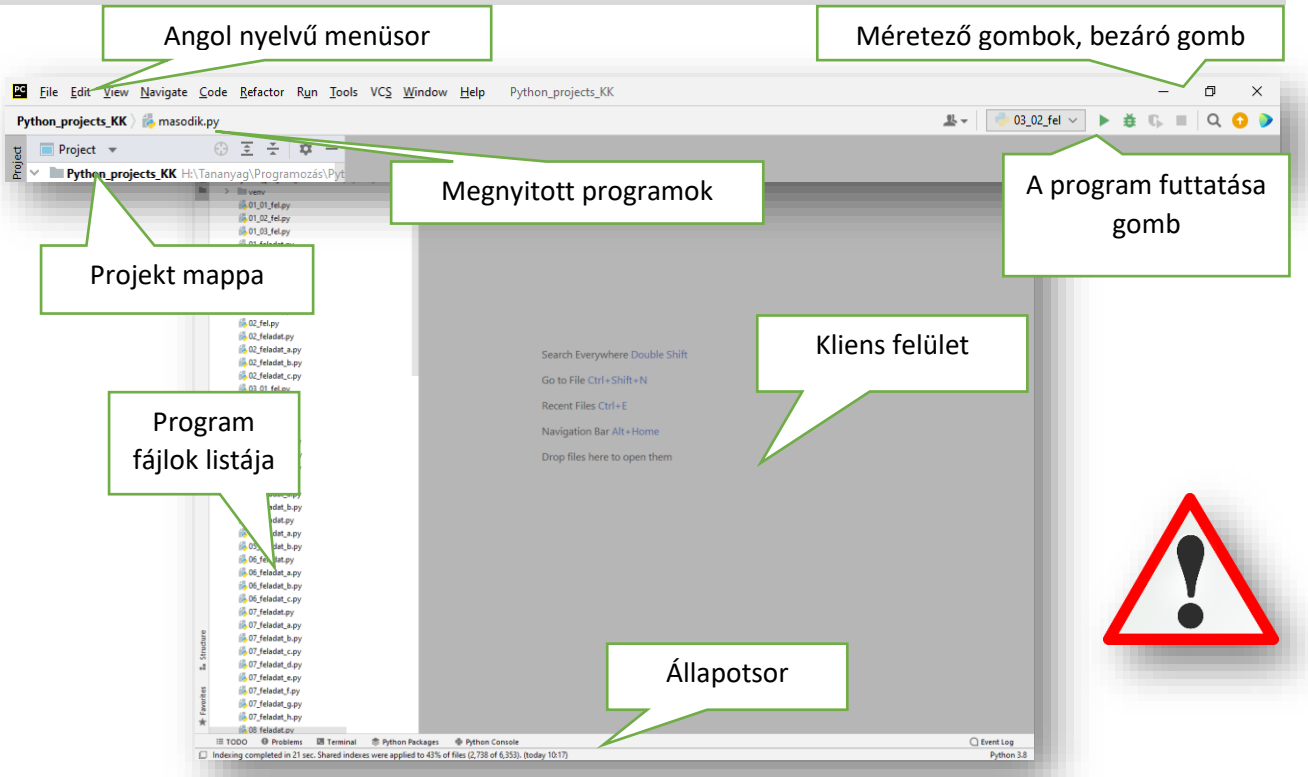


- A program elindulása után állítsunk be egy nekünk megfelelő témát, a jobb olvashatóság, kezelhetőség miatt!
  A File menü / Settings menüpontjában válasszuk ki az Appearance & Bihavior alpontjából a Theme: legördülő menüből, a Windows 10 Light témát!





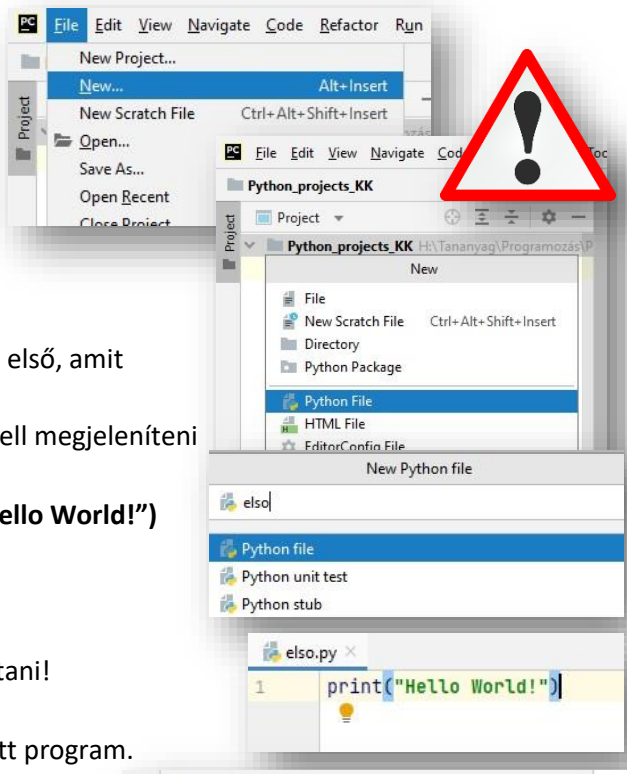
A PYCHARM PROGRAM FELÉPÍTÉSE

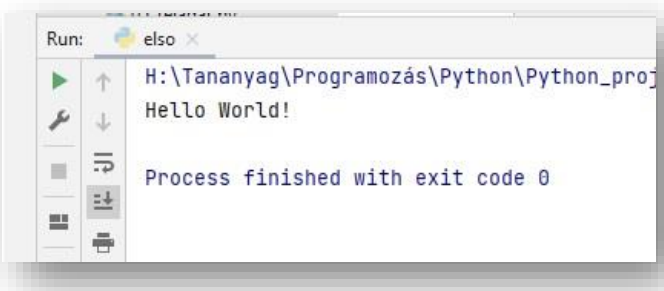


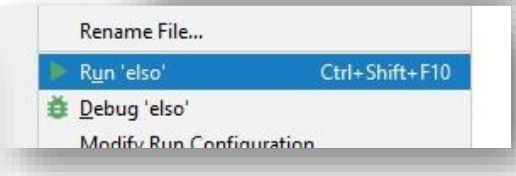
AZ ELSŐ PROGRAM ELKÉSZÍTÉSE

Hozzuk létre az első programunkat

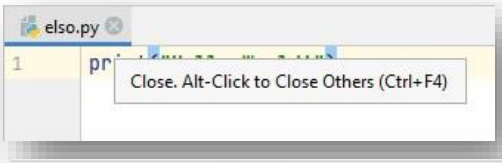
- **File / New**… menü kiválasztása az első
- **Python file** kiválasztás a második
- **Adjuk** a programunknak **nevet**! Pl.: „elso”
- Mint minden programozási nyelv tanulásánál az első, amit **kiíratunk a képernyőre** a „Hello World” szöveg.
  Ebben az esetben is nézzük meg, hogy hogyan kell megjeleníteni a szöveget!
- Gépeljük be a programozási felületre a **print(„Hello World!”)** parancsot!
- A **futtatáshoz** fent a program nevének „fülén”
  jobb egér „Run ’elso’ menüpontot kell kiválasztani!
- Végül megjelenik **lent a képernyő alján** a lefutott program.



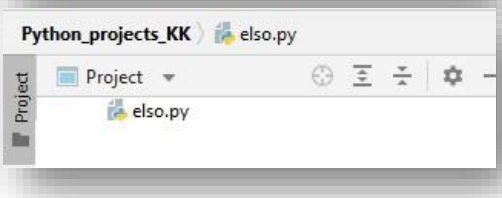




Ha az éppen megnyitott programot/kódot **be akarjuk zárni**, akkor fent a megnyitott ”fülek” közül bezárjuk az „X” gombbal.



Ha **meg akarunk nyitni**, egy programot/kódot, akkor a bal oldali listából kiválasztjuk.

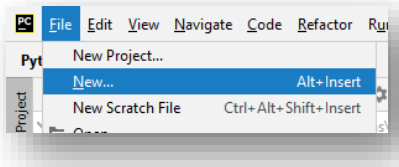


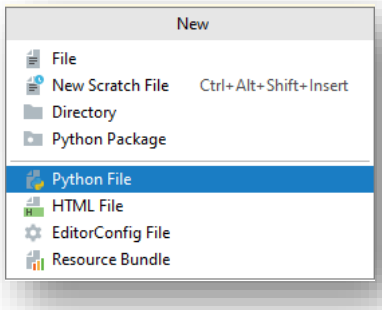
01. ALAPMŰVELETEK

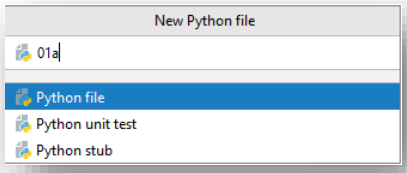
Nézzük meg, hogy hogyan használjuk az **egyszerű matematikai műveletek**et a python programozáskor. A parancsértelmező nagyon könnyen használható, csak be kell írni egyszerűen a „print()” utasításba a megszokott műveleteket.

(01a.py)

Hozzunk létre egy új python fájlt, a neve legyen 01a!

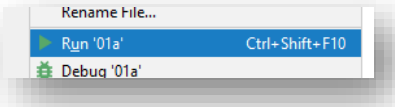


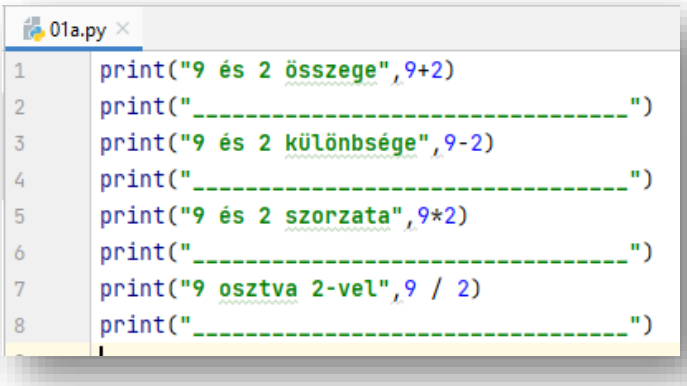


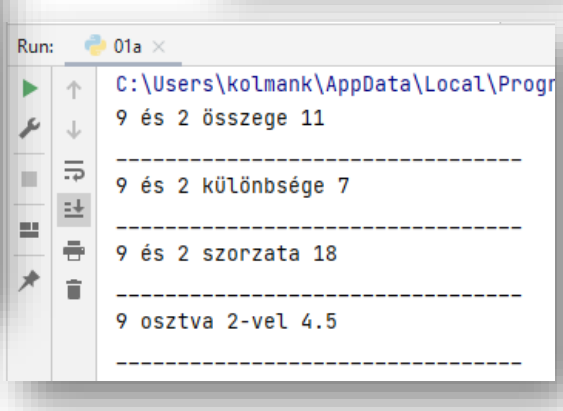


Ebben a feladatban az **összeadás, kivonás, szorzás és osztás műveletei**t próbáljuk ki!

- Gépeld be a képen látható utasításokat, majd futtasd a programot!
- A **kiíratáshoz a ”print()” parancs**ot úgy használjuk, hogy **ha szöveget akarunk megjeleníteni, akkor a zárójelben idézőjelek („szöveg”), vagy aposztrófok (’szöveg’) közé írjuk a karaktereket**!
- **Ha szöveget szeretnénk kiíratni és mellette még számolandó műveletet is elhelyezni akkor vesszővel kell elválasztani egymástól őket!**
- **A képletekben a matematikában használt módon alkalmazzuk a műveleti jeleket! (+, -, *, /)**
- A sorok között vonalakat húzunk, melyet elég egyszer begépelni, utána csak másolunk! Használhatjuk a kijelölés után a Ctrl+C; Ctrl+V billentyűkombinációkat!
- A jobb oldali képen láthatod a futtatott program eredményét.



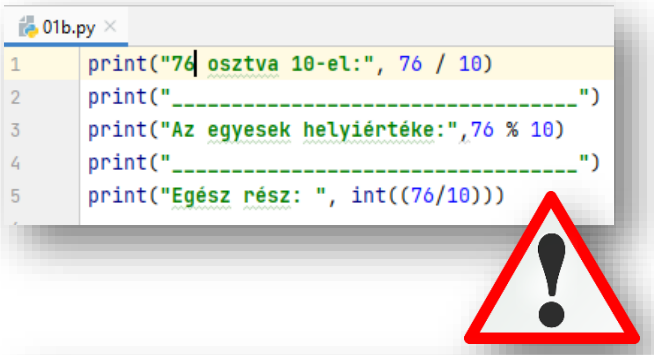


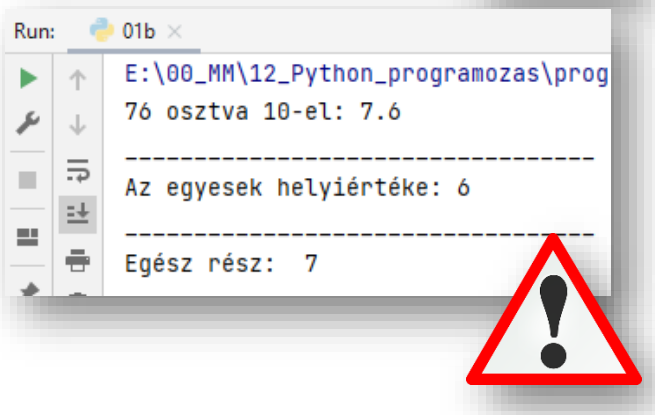


(01b.py)

Hozzunk létre egy új python fájlt, a neve legyen 01b!
Ebben a feladatban az osztásnál használatos **egész rész meghatározása (div) és a maradék meghatározására (mod)** fogunk programot készíteni.
Írassuk ki a 75 egyeseinek helyén található számjegyet, majd a tízesek helyén találhatót is!
Gépeljük be a programot a mint alapján!

- Ha két számot elosztunk egymással, akkor a várható módon tizedesre pontosan kiírja a program.
- (76/10)=7,6
- Majd a szokott módon húzzunk egy vonalat!
- A 76 egyeseinek helyiértékén 6 áll. Ezt a számot úgy kapjuk meg képlettel, hogy a 76-ot elosztjuk maradékosan 10-el, és így megkapjuk a maradékot. Ebben az esetben a 6-ot!
- **Az osztási maradék műveleti jele a %.** Tehát a két szám közé írunk egy % karaktert a mint alapján!
- A parancsértelmező soraiban látjuk az eredményt!
- Ha az **egész részre van szükségünk** akkor az „integer” (egész) szóból származó **int() utasítás**t fogjuk használni! (Az Excel programban is találkoztunk az int() függvénykénnyel. Itt is ugyanúgy működik az int(művelet)). Tehát ebben az esetben a futtatott programban a 7-es szám jelenik meg. (Vagy az int() helyett írhatunk // karaktereket is. Így is működik. 75 // 10)



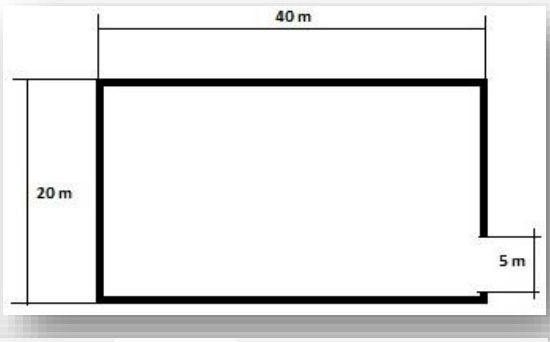


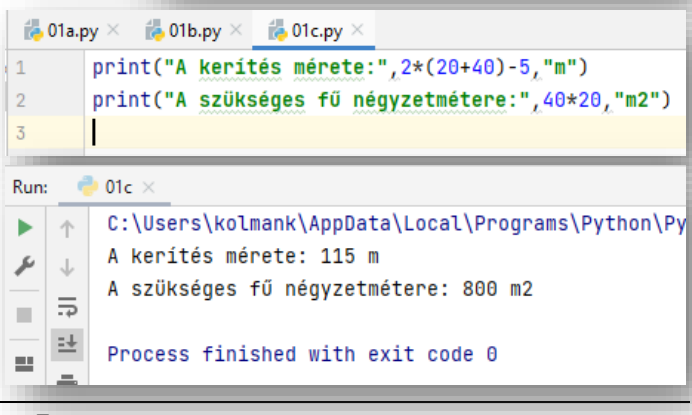
(01c.py)

Hozzunk létre egy új python fájlt, a neve legyen 01c!

Vásároltál egy telket, amely 20 méter széles, 40 méter hosszú és van rajta egy 5 méteres kapu.
Szeretnéd körbe keríteni. Számold ki, hogy mekkora, hány méter kerítésre lesz szükségünk!
Szeretnéd fűmaggal bevetni az egész kertet. Mekkora területre kell fűmagot vásárolnod?
A feladatot úgy készítsd el, ahogy a képen látod!

- A print() utasítás végén szerepeljenek a mértékegységek! Tehát egy újabb vesszővel válaszd el az idézőjelek közötti szöveget!
- A kerület képlete: 2*(a*b)
- Ebben az esetben a kapu 5 méterét levonjuk. 2*(20*40)-5
- A terület képlete: a*b
- Ebben az esetben: 20*40

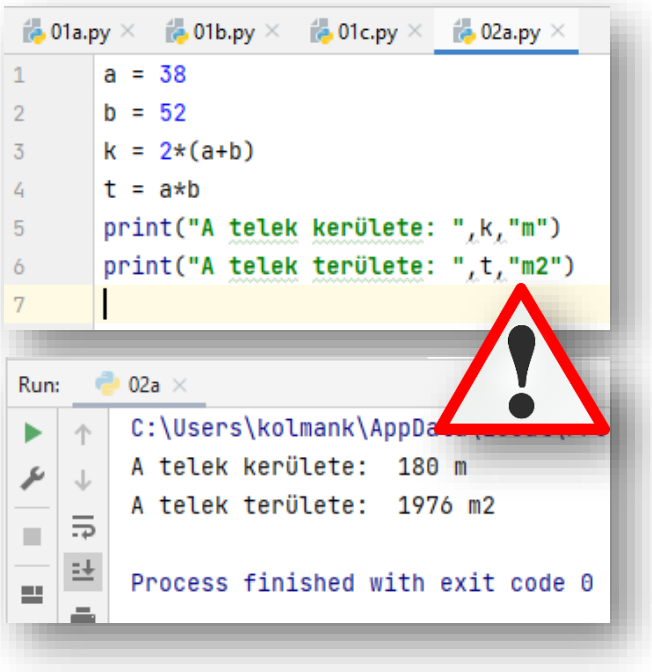




02. VÁLOZÓK HASZNÁLATA

Változók használata:
A programozásban nagyon fontos szerepük van a változóknak. Vannak olyan **értékek, amelyeket többször felhasználunk** a programunk során, vagy **mi adunk értékeket egyes esetekben**. Ezeknek a **változóknak „neveket" adunk és a programban ezekkel „hivatkozunk" rájuk**. Tetszőleges nevet adhatunk a változóknak, (lehetőleg olyat, amit könnyen megjegyzünk) majd = jel segítségével értéket adhatunk nekik. Ezt úgy képzeljük el, hogy van egy kamrai polc rendszer, amit felcímkézünk, majd a polcokra különböző dolgokat teszünk. **Egyszerű változók fajtái: egész szám, valós szám, karakterlánc, logikai**

(02a.py)
Hozzunk létre egy új python fájlt, a neve legyen 02a!

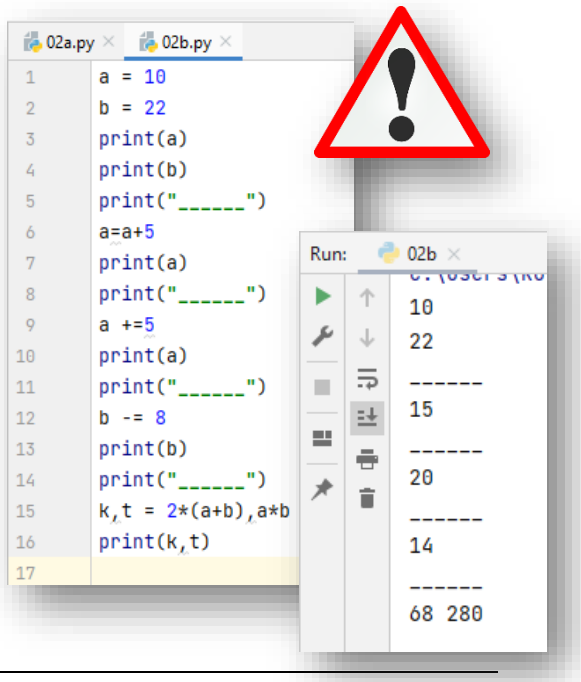


Az előző feladatban lévő „telek"-hez visszatérve, számoljuk ki a kerületét és a területét!

- A program elején adjuk értéket az „a" és a „b" oldalnak a minta szerint! a=38, b=52
- Majd egy „k" változóba számoljuk ki képlettel a kerületet!
- Aztán egy „t" változóba képlettel számoljuk ki a területet, úgy, hogy a program elején definiált a és b változókat használjuk!
- A változókat csak egyszerűen begépeljük a felhasználni kívánt helyen! (Pl.: print(k); k=2(a*b))
- Arra mindig figyelni kell, hogy csak olyan változót használhatunk fel képletben, amit előbbre definiáltunk!
- A k változóba és a t változóba belekerül a kiszámított érték, amit a következő két sorban a minta szerint kiíratunk!

Zárjuk be a megnyitott 01-es feladatokat! Csak a 02a.py maradjon megnyitva!

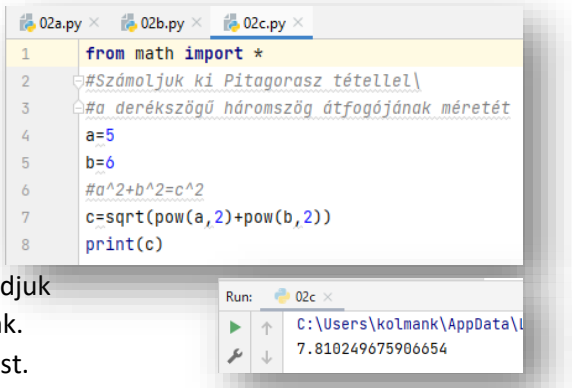
(02b.py)
Hozzunk létre egy új python fájlt, a neve legyen 02b!



Ebben a feladatban egy **változó értékének növelését illetve csökkentését** fogjuk kipróbálni. Gépeljük be az utasításokat!

- Az a = a+5 utasítás kicsit megzavarhatja a kezdő programozót, mert ez matematikailag nem helyes. De ebben az esetben ez egy új értéket ad a változónak. Azt mondjuk, hogy az egyenlőség jel az azt jelenti, hogy „**legyen egyenlő**"! Tehát az a értéke legyen egyenlő az eredeti a érték plusz 5-el! Ebben az esetben az értéke 10, akkor e művelet után 15 lesz.
- Tehát nem a bal és a jobb oldal összehasonlítására kell gondolnunk. Erre majd a későbbiekben az == jel fog szolgálni.
- Egy utasításban több értéket is adhatunk, ha a változókat és az értékéül kapott kifejezéseket vesszővel elválasztjuk egymástól.

(02c.py)
Hozzunk létre egy új python fájlt, a neve legyen 02c!



A feladatban számoljuk ki egy **derékszögű háromszög átfogójának méretét**, ha meg van adva a két befogó!

- A feladatot Pitagorasz tétellel oldjuk meg. $\mathbf{a^2+b^2=c^2}$
- Ahhoz, hogy a négyzetre emelés és a gyökvonás függvényt tudjuk használni egy úgynevezett függvénykönyvtárt kell betöltenünk.
- Ehhez be kell írni az első sorba a:**from math import *** utasítást.
- Ha **megjegyzéseket** szeretnénk írni az utasítások mellé akkor a **„#" karakter** beírása után megtehetjük!
- Mint tudjuk egy **utasítás**nak egy sorban kell lennie. Ha mégis **több sorba** akarnánk **írn**i, akkor a **„\" karaktert kell használni**.
- Ennek a feladatnak a megoldásánaál használnunk kell a **gyökvonás sqrt() és a hatványozás pow(alap,kitevő)** függvényeket!
- Gépeljük be és értelmezzük az utasításokat!

Áttekintés és fontos kiegészítések az eddig tanultakhoz:

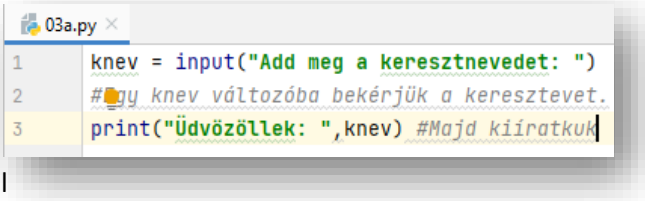
- A parancsokat parancssorokba írjuk, melyek egymás után hajtódnak végre a futtatás során.
- Egymás alá írt utasítások egységét programnak hívjuk.
- Parancssorok begépelésekor minden sor végén entert kell nyomi. Ezzel jelezzük, hogy befejeztük a parancs begépelését, jöhet a végrehajtás.
- Minden utasítást új sorban kell kezdeni.
- Ha egy utasítás nagyon hosszú több sorban szeretnénk megjeleníteni, akkor a "\" jelet kell használni. (Nem per (/) jel!
- A sorok elején jelentősége van a szóközöknek. Nem mindegy, hogy hol kezdődik az utasítás! Későbbiekben az összetartozó részeket beljebb fogjuk kezdeni.
- Az utasításokon belül szabadon bánhatunk a szóközök használatával!
- A karakterláncokat, (például a print parancsnál) írhatjuk idézőjelek közé vagy aposztrófok közé! De persze nem lehet keverni őket!
- A változó a memória egy darabja, amire a nevével hivatkozhatunk. A nevének az angol abc egyik betűjével kell kezdődnie. A nevének további részében lehet az angol abc betűi, számok és aláhúzás. Nem egyezhet meg a neve egy lefoglalt utasítás nevével (pl.: nem lehet „print" a változó neve)!
- A változókon végrehajtott legfontosabb művelet az értékadás. A x=50 azt jelenti, hogy „az x legyen egyenlő 50-el".
- Több változón egyszerre is adhatunk értéket: x,y,z=10,20,30
- Amikor értéket adunk egy változónak, azzal eldöntjük a típusát. (Úgy mind az Access adatbázis-kezelésnél.)
- A változó típusai lehetnek: karakterlánc (string); egész számok, valós számok, valamint logikai értékek (True, False)
- A szám változóknál az érték növelésére, csökkentésére, változtatására működnek a+=, -=, *= parancsok. Tehát a bal oldali változó értéket növelik, csökkentik, vagy éppen szorozzák a jobb oldalon megadott értékkel.
- Ha hibaüzenet jelenik meg piros színnel, ez figyelmeztet, hogy valami el lett gépelve, vagy kimaradt valami. A parancsértelmező megpróbálja jelezin, hogy van a hiba. Ahol villog a kurzor ott kell keresni a hibát. Illetve szövegesen is kiírja, hogy mit hibázhattunk el.
- Ha **megjegyzéseket** szeretnénk hozzáfűzni egyes sorokhoz a python programozásakor, akkor a **„#" karaktert használjuk**.

03. ADATBEKÉRÉS

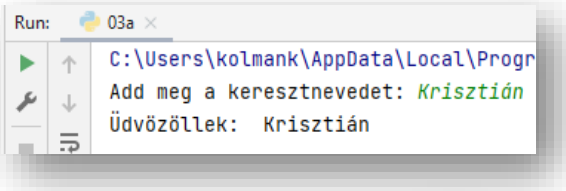
Adatbekérés:
Eddig minden esetben a program elején fixen megaduk a változóknak az értékeket. De legtöbbször arra van szükségünk, hogy **a felhasználó adja meg a változók értékét**. Mindezt anélkül, hogy a programunkat át kelljen szerkeszteni. A következőkben különböző típusú adatokat fogunk bekérni, majd azzal műveleteket végrehajtani.

(03a.py)
Hozzunk létre egy új python fájlt, a neve legyen 03a!

Készítsünk programot, ami bekéri a keresztnevedet, majd üdvözöl téged!



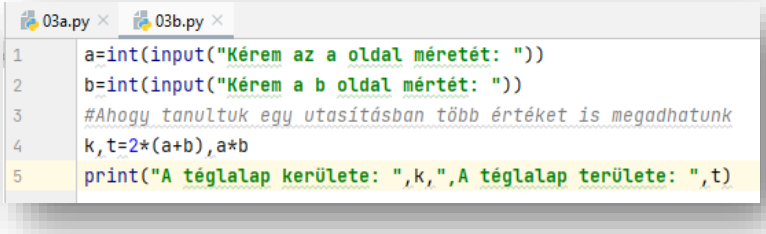
Az adatbekérést az input() utasítással tudjuk megtenni!
Először adunk a változónak egy nevet, majd egyenlőség jel után beírjuk az input utasítást két zárójellel!
A zárójelek közé idézőjelekben beírunk egy szöveget, melyben a bekért adatra kérdezünk rá!
karakter után írhatunk rövid magyarázatot, hogy mit csinál éppen a program!
Kiíratni pedig az eddigiekben tanultak alapján lehet.
Gépeld be az utasításokat a minta alapján!



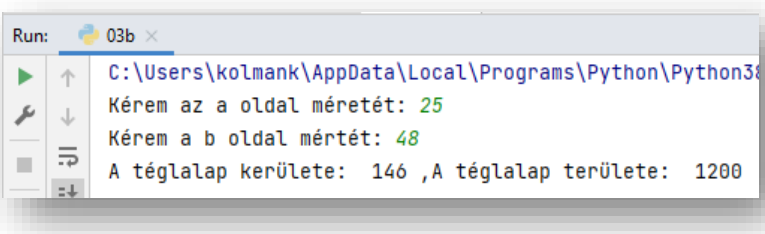
Típusátalakítás, konverzió:
Tapasztaltuk, hogy különböző adattípusok vannak. Az előzőleg tanult input utasítás szöveg (string) karakterláncokat olvas be, de a karakterláncokkal nem lehet matematikai műveleteket végezni. Nem lehet összeadni, szorozni. Ezért szükségünk van arra, hogy átalakítsuk, átkonvertáljuk őket egész vagy valós számmá!
A karakterláncból egészet az int(), valós számot a float() utasítással tudunk konvertálni.

(03b.py)
Hozzunk létre egy új python fájlt, a neve legyen 03b!

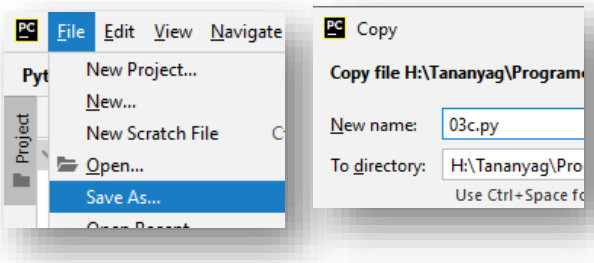
A feladat ismerős, egy téglalap kerületét, területét kell kiszámolni úgy, hogy az **a és b oldalt a felhasználótól kérjük be**!



Az első sor elején adjuk meg a változó nevét a szokásos módon („a”), majd mivel tudom hogy szövegként fogom bekérni a változót, ezért az int() függvénnyel indítva kérjük be a változót input()-al a minta alapján!
Ugyan ezt tegyük meg a b oldallal is!
Majd egy sorban több utasítást megadva számoljuk ki a kerületet és a területet!
Végül írassuk ki az eredményeket a minta alapján!

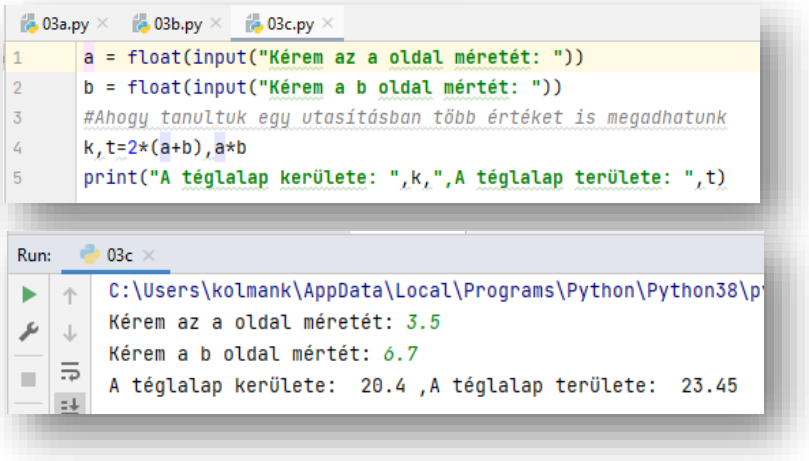


(03c.py)
Az előző programot mentsük el másként 03c.py néven!
File/Save as.. -> 03c.py



Ha fel lehet használni egy régebbi programot egy feladathoz, akkor **mentsük el másként, és változtassuk meg a tartalmát**! Így nem kell az egészet újból beírni.

- Most csak annyit kell tennünk, hogy amikor bekéri az „a” és a „b” oldalakat, ne csak egész számot lehessen beírni, hanem törtet is; az int() utasítást kell kicserélni a float() utasításra!
- FONTOS: A feladat tesztelésénél, amikor beírjuk a számokat akkor a tizedesvessző (,) helyett, pontot (.) kell beírnunk!



Változók cseréje

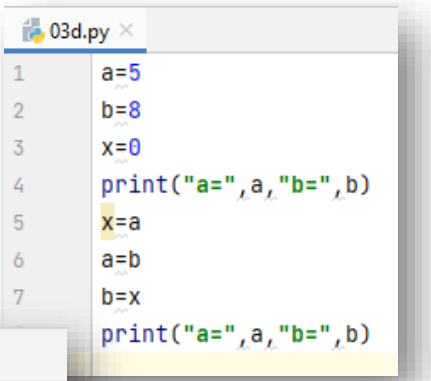
Az egyik tipikus programozási feladat, amikor **két változónak a tartalmát fel kell cserélni**. Tehát bekérünk egy „a” és egy ”b” változót, majd kiíratjuk az eredeti értékeket. Aztán cseréljük meg a két értéket és írjuk ki újból a már megcserélt számokat!

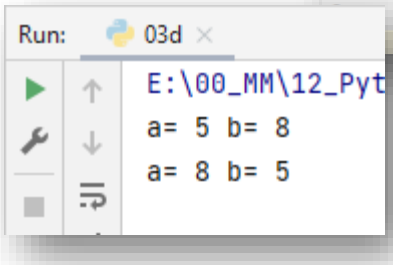
A feladat megoldásához szükségünk lesz egy köztes „x” változóra, melyben ideiglenesen tudjuk tárolni az értékeket. Nézzük a lépéseket: ha az eredeti változók értéke: a=5, b=8, x=0

- **x ⇨ a** a=5, b=8, x=5
- **a ⇨ b** a=8, b=8, x=5
- **b ⇨ x** a=8, b=5, x=5

(03d.py)
Hozzunk létre egy új python fájlt, a neve legyen 03d!

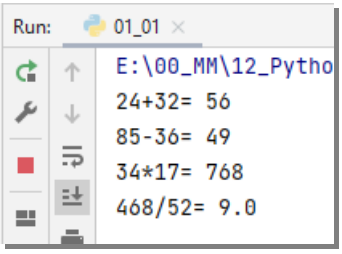
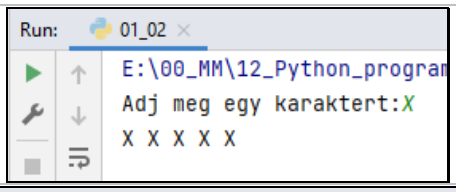
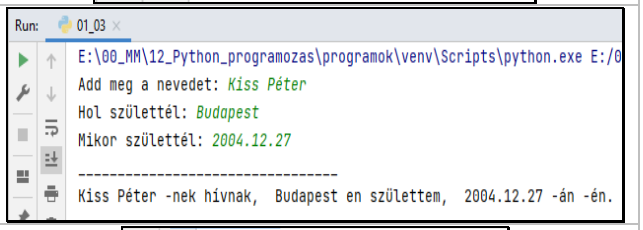
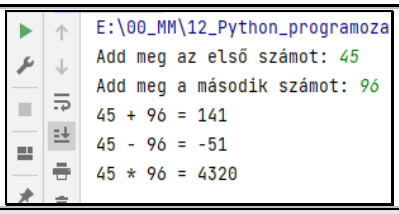
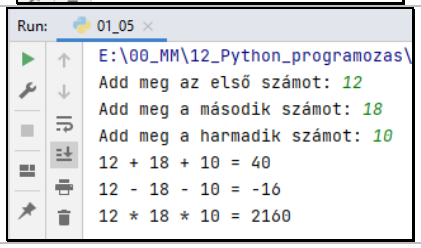
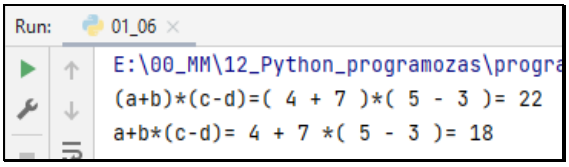
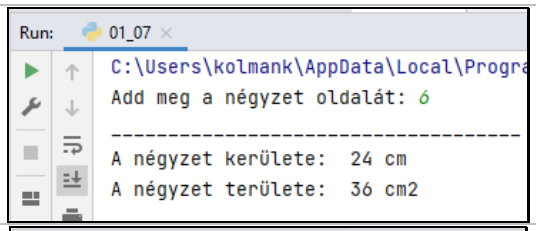
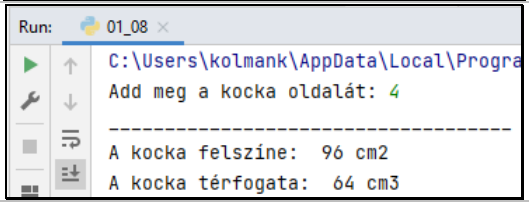
- Valósítsuk meg az előzőekben leírt példát!
- A program elején adjuk meg az a,b,x értékét!
- Írasd ki az eredeti a és b értéket!
- Majd cseréljük meg az a és a b értékét a tanult módon!
- Majd újból írassuk ki a megcserélt a és b-t!



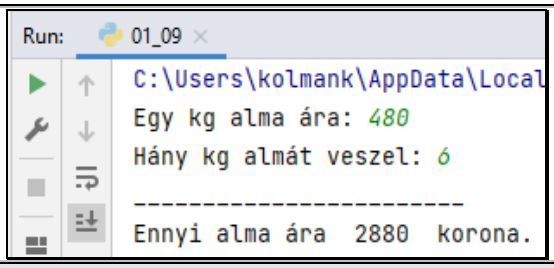


GYAKORLÓ FELADATOK (01-03. témakör)

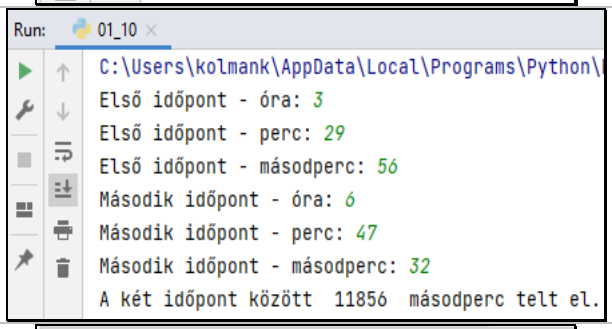
Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (01_01.py) Írasd ki a következő képleteket a képernyőre, úgy hogy az eredmény számolva legyen! 24+32=56 85-36=49 34*17=578 468/52=9	
2. (01_02.py) Írj egy programot, amely bekér egy karaktert és kiíratja egymás mellé ötször!	
3. (01_03.py) Készítsed el a képen látható adatbekérő kérdőívet, majd írasd ki az alatta lévő mondatot!	
4. (01_04.py) Írjál egy programot, melyben bekérsz két számot! Aztán a számokat összeadod, kivonod és összeszorzod a minta alapján!	
5. (01_05.py) Bővítsed ki az előző programot úgy, hogy három számot kérjen be, és azzal a három számmal végezze el a műveleteket!	
6. (01_06.py) Készítsed el a következő két képletet a minta alapján az előre deklarált számokkal! a=4 b=7 c=5 d=3	
7. (01_07.py) Készíts programot, amely bekéri egy négyzet oldalának egységnyi méretét, majd kiszámolja annak kerületét és területét!	
8. (01_08.py) Alakítsad át az előző programot úgy, hogy az egy kocka felszínét és térfogatát számolja ki!	

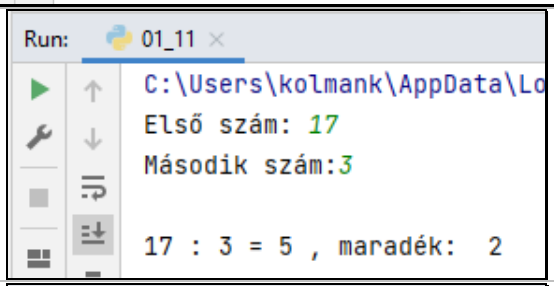
9. (01_09.py)
Almát szeretnénk vásárolni. Készíts egy programot, amelyben először bekéri egy kilogramm alma árát, majd azt, hogy hány kilogramm almát szeretnénk venni. Végül a program számolja ki nekünk, hogy menni koronát kell fizetnünk.



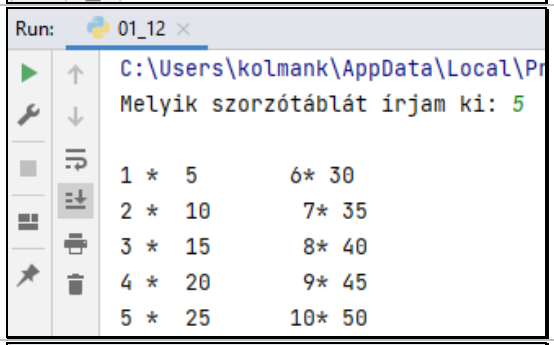
10. (01-10.py)
Készítsél egy programot, melyben bekérünk két (egy napon belüli) időpontot. Majd add meg a két időpont közötti másodpercek számát!
Időpont formátuma: óra:perc:másodperc
Az első sorban kell a `from math import *`
Szükség van a : `abs(kulonbseg)` függvényre



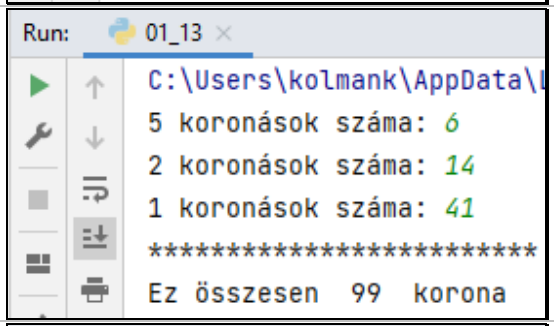
11. (01_11.py)
Írjál egy programot, amely bekér két egész számot, majd kiírja a két szám egész hányadosát és maradékát az alábbi formában!
A program az adatok beolvasása után hagyjon ki egy üres sort!



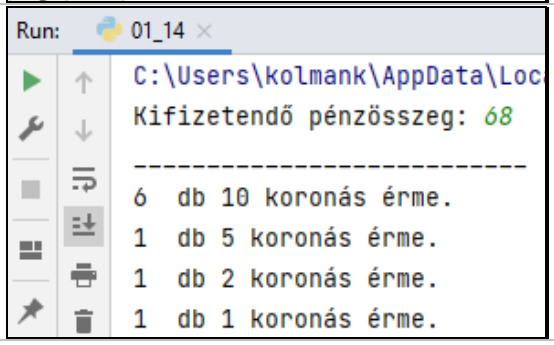
12. (01_12.py)
Írj egy programot, amely bekér egy egész számot, aztán készítsen egy egyszerű szorzótáblát a képernyőkép alapján!
A szám bekérése után hagyjon ki egy üres sort!



13. (01_13.py)
Kérjünk be három egész számot! Ezek a számok feleljenek meg sorban 5 koronásnak, 2 koronásnak, 1 koronásnak!
Majd számoljuk ki, hogy összesen hány koronánk van!



14. (01_14.py)
Készíts programot, amely bekér egy bizonyos pénzösszeget! A szám legyen 1-100 közötti!)
Majd határozd meg, hogy hogyan tudjuk a legkevesebb 10, 5, 2, illetve 1 koronással kifizetni!



15. (01_15.py)

Készíts programot, amely bekér három nem egész számot (x,y,z) majd készítsed el a következő képleteket!

$x^2+y^2+z^2=$

$x^y+z^3=$

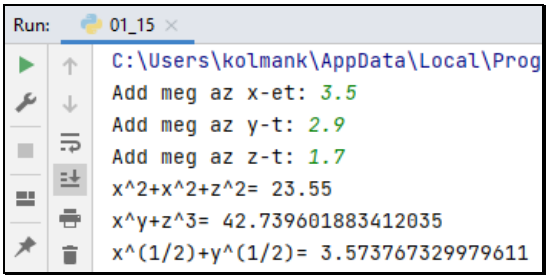
$\sqrt{x}+\sqrt{y}=$

A feladat megoldásához, a függvények használatához az első sorba gépeld be:

```
from math import *
```

szöveget!

A matematikai függvények használatához szükséges előzőleg be kell tölteni a „math" függvénykönyvtárat!



16. (01_16.py)

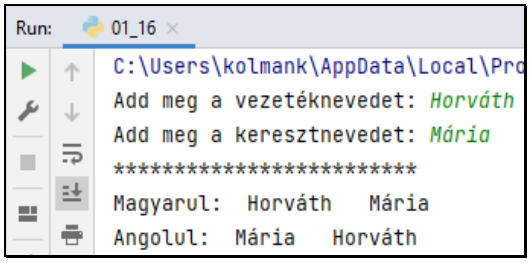
Készíts programot melynek az elején bekéri egy vnev és egy knev változóba a vezetéknevedet és keresztnevedet!

Majd írasd ki, hogy:

„Magyarul: Vezetéknév Keresztnév

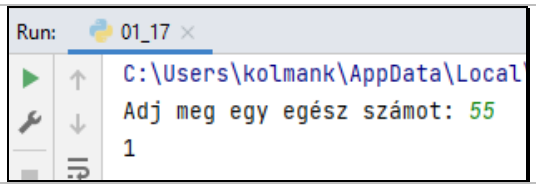
Angolul: Keresztnév Vezetéknév"

A programot úgy készítsed el, hogy a változókat fizikailag cserélje meg!



17. (01_17.py)

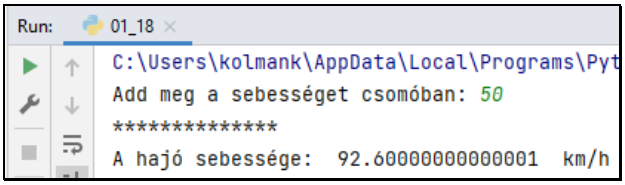
Kérjél be egy egész számot, majd írjon ki 0-t ha páros a szám, írjon ki 1-est, ha páratlan a szám!



18. (01_18.py)

A csomó a hajózás területén használt sebesség-mértékegység. Egy nemzetközi csomó alatt egy tengeri mérföld óránkénti sebességet értünk. Ez pontosan 1,852 km/h.

Készíts egy olyan programot, amelybe ha beírsz egy számot, ami csomóban van, akkor kiírja, hogy az mennyi km/h!

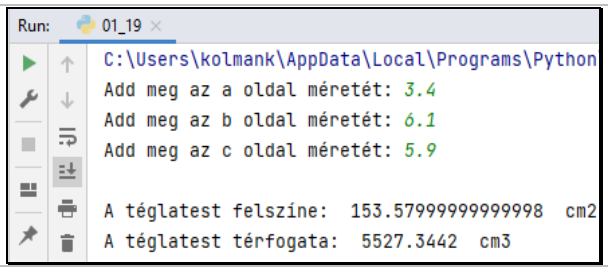


19. (01_19.py)

Számold ki egy téglatest felszínét és térfogatát!

Valós számokat várjon a program!

Az adatok bekérés után legyen egy üres sor!



20. (01_20.py)

Írjál programot, amely bekéri a mai dátumot és azt, hogy mikor születtél!

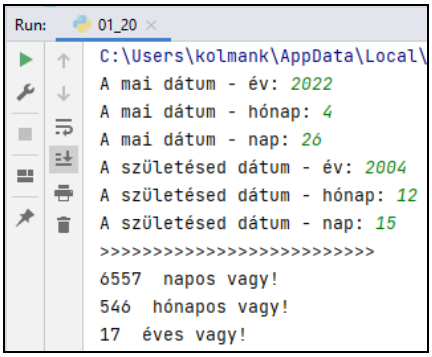
Számolja ki a program, hogy hány napos vagy!

Hány hónapos vagy!

Hány éves vagy!

(Ebben a programban nem számítanak a szökőévek!)

A feladatot a képernyőkép alapján készítsed el!



04. FORMÁZOTT KIÍRATÁS

Az eddigi tananyag során nem alkalmaztunk **sortörés**, nem használtunk **azonos sorban kiíratás**t különböző utasításoknál! Egyébként ezeket hívjuk **tagolás**nak.
Eddig amikor kiírattunk a képernyőre eredményeket, nem **formáz**tuk, nem adtuk meg, hogy milyen formátumban jelenítse meg az adott számot! Bonyolultabb esetekben a formázott adat kiíratást célszerű jelölőkkel (markarekkel) elkészíteni. Ezek lesznek a **„%”-jelek**.
Ezek használatát legegyszerűbben példákon keresztül lehet megérteni!

(04a.py)
Nyisd meg a 03b.py programot és mentsd el másként 04a.py néven!

Első lépésként alakítsd át a minta alapján, hogy ez legyen a kiindulási helyzet!

```
03b.py
1  a=int(input("Kérem az a oldal méretét: "))
2  b=int(input("Kérem a b oldal mértét: "))
3  k,t=2*(a+b),a*b
4  print("A téglalap kerülete: ",k)
5  print("A téglalap területe: ",t)
```

```
Run: 03b
C:\Users\kolmank\AppData\Loca
Kérem az a oldal méretét: 6
Kérem a b oldal mértét: 9
A téglalap kerülete:  30
A téglalap területe:  54
```

A célunk az, hogy úgy módosítsuk a programunkat, hogy a jobb oldali képen látható módon jelenjen meg az eredmény! Úgy változtassuk meg, hogy külön print utasítás legyen „A téglalap kerülete: „, valamint a 30! Viszont egy utasítás legyen, de külön sorban „A téglalap területe: ” és az 54!

```
Run: 04a
C:\Users\kolmank\AppData\Loca
Kérem az a oldal méretét: 6
Kérem a b oldal mértét: 9
A téglalap kerülete:  30
A téglalap területe:
54
```

Tagolt kiíratás

Tehát, ha két külön utasítást szeretnénk végrehajtani, amit **külön sorba írtunk, de** azt szeretnénk, hogy **egy sorban legyenek megjelenítve**, akkor az első sor utasításának végére vesszővel elválasztva írjuk be az **end=’ ’** utasítást!

```
04a.py
1  a=int(input("Kérem az a oldal méretét: "))
2  b=int(input("Kérem a b oldal mértét: "))
3  k,t=2*(a+b),a*b
4  print("A téglalap kerülete: ",end=' ')
5  print(k)
6  print("A téglalap területe: ",t, sep='\n')
```

Ha viszont, **szeretném külön sorba tagolni** az utasítás kiíratását, akkor szintén vesszővel elválasztva írjuk az utasítás végére a **sep=’\n’** utasítást!

```
04a.py
1  a=int(input("Kérem az a oldal méretét: "))
2  b=int(input("Kérem a b oldal mértét: "))
3  k,t=2*(a+b),a*b
4  print("A téglalap kerülete: ",end=' ')
5  print(k,end=' ')
6  print(k)
7  print("A téglalap területe: ",t,t,t,t, sep='\n')
8
```

```
Run: 04a
C:\Users\kolmank\AppData\Local
Kérem az a oldal méretét: 6
Kérem a b oldal mértét: 4
A téglalap kerülete:  20 20
A téglalap területe:
24
24
24
24
```

- Ha még jobban kibővítjük a programunkat, akkor biztosan megértjük, hogy hogyan működnek ezek a tagolások!
- Az end=’ ’ utasítást mindig be kell írni az utasítás végére!
- A sep=’\n’ -nél pedig minden vesszővel elválasztott tagot külön sorba ír!

Jelölők használata

A **formázott adatkiírás**t célszerű jelölőkkel megoldani. A jelölők százalék jellel kezdődnek, beírhatók a szövegbe, és a print utasítás egy későbbi részén elegendő megadni az értéküket.
Általános jelölő: **%s**, amelyeket **karakterlánc**ok helyettesítésére használunk; az **egész számok**nál **%d**; a **valós számok**nál **%f**-et adunk meg. Ezek megértéséhez szintén nézzünk példákat!

(04b.py)
Hozzunk létre egy új python fájlt, a neve legyen 04b.py!

Készítsünk programot, melyben bekérünk két közönséges törtet, majd megjelenítjük egyszerűsítés nélkül a szorzatukat, a következő formátumban: 15/11 * 5/3 = 75/33

A változók bekérése a már ismert módon történik!

04b.py

```
1  sz1=int(input("Kérem az első tört számlálóját: "))
2  n1=int(input("Kérem az első tört nevezőjét: "))
3  sz2=int(input("Kérem a második tört számlálóját: "))
4  n2=int(input("Kérem amásodik tört nevezőjét: "))
5  print("%d/%d * %d/%d = %d/%d" % (sz1,n1,sz2,n2,sz1*sz2,n1*n2))
```

Run: 04b

```
C:\Users\kolmank\AppData\Local\Progra
Kérem az első tört számlálóját: 16
Kérem az első tört nevezőjét: 17
Kérem a második tört számlálóját: 9
Kérem amásodik tört nevezőjét: 8
16/17 * 9/8 = 144/136
```

- A jelölők használatára példát az utolsó sorban látunk.
- A print utasítás zárójelben lévő dolgait három részre bonthatjuk.
- Az első harmadban az idézőjelek között, ahova az egész számok fognak kerülni, ott a %d jelölőt használjuk.
- A második harmadba csak simán beírunk egy % karaktert!
- AZ utolsó harmadba pedig zárójelek közé beírjuk a változókat sorban egymás után, ahogy egymás után következnek az első, idézőjeles részben!
- Vizsgáld meg alaposan a program képernyőképét és értelmezd!

Tizedes törtek megjelenítése adott pontossággal

A valós számoknál a tizedes helyeknek megfelelő megjelenítést a következő módon készítjük el.
A **%.2f** azt jelenti, hogy két tizedes pontossággal jelenítjük meg a valós számot.

(04c.py)
Hozzunk létre egy új python fájlt, a neve legyen 04c.py!

A feladat az, hogy olvasson be a program egy alapszámot és egy értéket! A kérdés az az, hogy az alapnak hány százaléka az érték?

04c.py

```
1  a = int(input("Kérem az alapot: "))
2  e = int(input("Kérem az értéket: "))
3  print("alap: %d, érték: %d , százalékláb: %.2f" % (a,e,e/a*100))
```

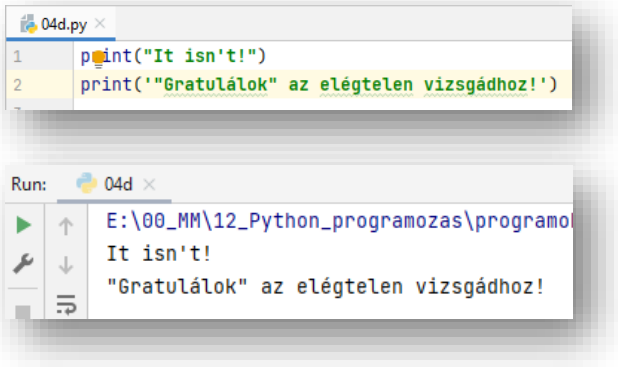
Run: 04c

```
C:\Users\kolmank\AppData\Local\Programs\Pyt
Kérem az alapot: 654
Kérem az értéket: 123
alap: 654, érték: 123 , százalékláb: 18.81
```

- Gépeljük be a programot!
- Majd értelmezzük a beírtakat!
- A print parancsban az egész számok helyén %d, a valós szám helyén a %.2f jelölő áll!
- Szintén három részre tagolhatjuk a print utasításban lévő parancsokat!
- Figyeljünk, hogy a % karakter ne maradjon ki!

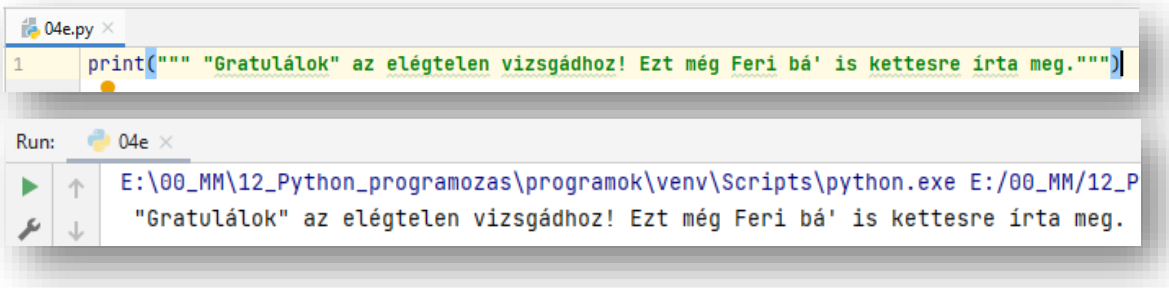
(04d.py)
Hozzunk létre egy új python fájlt, a neve legyen 04d.py!

Előfordulhat, amikor a kiírandó szövegünkben aposztrófokat vagy idézőjeleket kell használnunk. Az idézőjelek által közrefogott szöveg aposztrófot is tartalmazhat ("It isn't."), az aposztrófok által határolt szöveg pedig tartalmazhat idézőjelet is ('"Gratulálok" az elégtelen vizsgádhoz!').



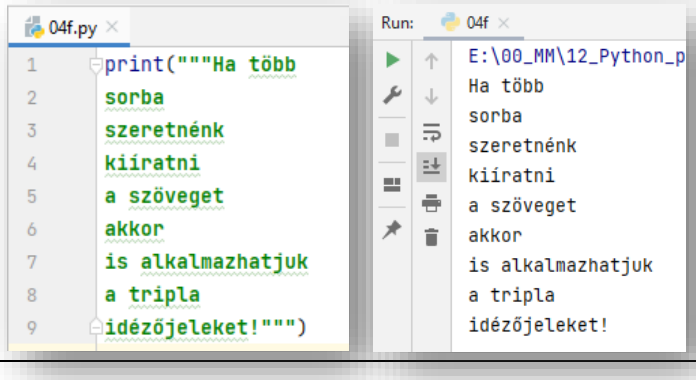
(04e.py)
Hozzunk létre egy új python fájlt, a neve legyen 04e.py!

Ha véletlenül olyan szöveget kell kiíratnunk, amelyben egyszerre szerepel aposztróf és idézőjel is, akkor a szövegünket tripla aposztróf, vagy idézőkelek közé kell íratni!



(04f.py)
Hozzunk létre egy új python fájlt, a neve legyen 04f.py!

A tripla idézőjelek és aposztrófok alkalmazásakor több sorba írathatjuk ki a szövegünket!



A Pythonban vannak úgynevezett **lefoglalt szavak**, amelyeket nem lehet felhasználni egyéb esetben, például nem adhatjuk változó neveknek. Vannak köztük olyanok, amelyeket már ismerünk és vannak olyanok, amelyekkel később fogunk megismerkedni. **Ezek a következők**:

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

05. VÉLETLEN SZÁMOK GENERÁLÁSA/HASZNÁLATA

A programozás feladatok nagyon sok hányadában van szükségünk **véletlen számok generálására**! Ha példát kell mondani, akkor általában a lottó sorsolás vagy a kockadobást szokták említeni.
Először az első sor(ok)ban meg kell adni a „math" függvénykönyvtár meghívásához hasonlóan, be kell gépelni a `from random import *` parancsot!
Python programozásban véletlen egész számokat a **randrange** utasítással készíthetünk. Az **utasítás után meg kell adni egy (n) számot**, mely akkor egy **0 és n-1 közötti véletlen számot generál**!
Egy 0 és 1 közötti véletlen számot a random utasítással generálhatunk! (A szám soha nem lehet 1!)
Ezekkel az utasításokkal generált számokkal műveleteket végeznetünk. Hozzá adhatunk számokat, kivonhatunk belőlük, szorozhatjuk, stb…
Tehát ezek alapján, ha **kockadobást** modelleznénk, akkor a randrange(6)-al csak egy 0-5 közötti számot hoznánk létre, de ha **randrange(6)+1**-et írnánk, akkor 1-6-ig generálna véletlen számot!

A **véletlen számok készítése nem tartozik a Python alaputasításai közé**, ezért itt is az import utasítást kell használni a program elején. Mégpedig a „**from random import** *" -ot kell begépelni! (A * azt jelenti, hogy minden hozzá tartozó függvényt elérhetővé tesz.)
Egyébként a randrange utasításnál meg lehet adni egy kezdő értéket, egy felső határt és hogy milyen lépésközzel ugorjon! `randrange(start, stop, step)`

(05a.py)
Hozzunk létre egy új python fájlt, a neve legyen 05a.py!

Készítsünk programot, melyben kettő darab (az előbb említett) kockadobást modellezed, majd utána két 0 és 1 közötti véletlen számot generálj!

- A program elején írjuk be az első sorba a `from random import *` utasítást!
- Generálj a feladatban leírt módon véletlen számokat a változókba!
- A kiírt számok formátumát a következő képen állítsd be: az első kettő egész, a második kettő valós, négy tized pontossággal legyen ábrázolva!
- A számok egymás mellett jelenjenek meg a minta szerint!

```
05a.py
1  from random import *
2  a = randrange(6)+1
3  b = randrange(6)+1
4  c = random()
5  d = random()
6  print( "%d, %d, %.4f, %.4f" % (a,b,c,d))

Run:  05a
   C:\Users\kolmank\AppData\Local\Prog
   5, 6, 0.3135, 0.9907

   Process finished with exit code 0
```

(05b.py)
Hozzunk létre egy új python fájlt, a neve legyen 05b.py!

Készítsünk programot, melyben generálunk öt olyan számot, ami 100 és 200 között van!
A számok 10-esével legyenek léptetve!

- A program elején írjuk be az első sorba a `from random import *` utasítást!
- Generálj a feladatban leírt módon véletlen számokat a változókba!
- Végül Írassuk ki a minta szerint egyszerűen egymás mellé a számokat!

```
05b.py
1  from random import *
2  sz1=randrange(100,200,10)
3  sz2=randrange(100,200,10)
4  sz3=randrange(100,200,10)
5  sz4=randrange(100,200,10)
6  sz5=randrange(100,200,10)
7  print(sz1,sz2,sz3,sz4,sz5)

Run:  05b
   C:\Users\kolmank\AppDa
   140 170 170 110 190
```

06. LOGIKAI VÁLTOZÓK HASZNÁLATA

A logikai változók értéke igaz (**True**) vagy hamis (**False**) lehet. Kötelezően nagy betűkkel használjuk! Egy logikai változót, egy eldöntendő kérdésre adott válasz tárolására használhatjuk.
Az Excel táblázatkezelő programnál tanult logikai függvények, az és, vagy, nem itt is használhatók.
A logikai értékeket adhatunk, közvetlen megadással: A=True; vagy kifejezés eredményeként: B=(C<D)
Nézzük át a függvények igazságtábláit!

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A v B
0	0	0
0	1	1
1	0	1
1	1	1

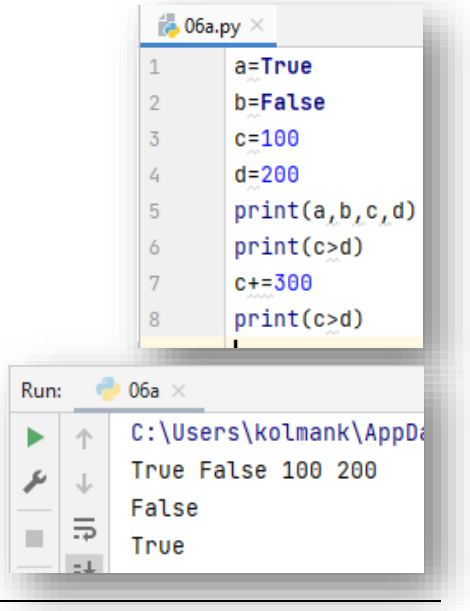
A	NEM(A)
0	1
1	0

- **Az ÉS függvény: Akkor igaz, ha minden állítás igaz.**
- **A VAGY függvény: Akkor igaz, ha legalább az egyik állítás igaz.**
- **A NEM függvény: Az ellenkezőjére változtatja az értéket!**

(06a.py)
Hozzunk létre egy új python fájlt, a neve legyen 06a.py!

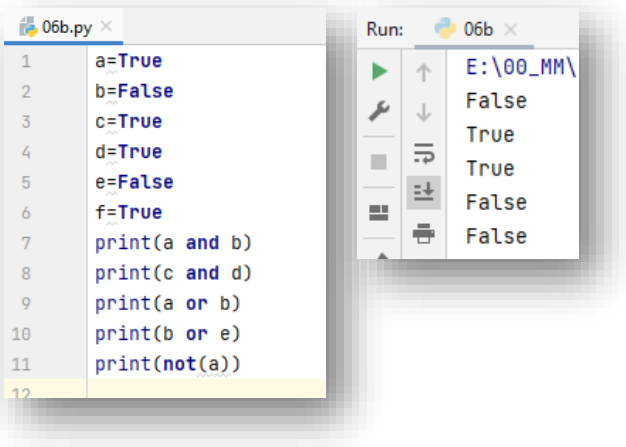
Próbáljuk ki a logikai utasításokat!

- Adjuk értéket a,b,c,d változóknak a minta alapján!
- Írassuk ki az eredeti értékeket!
- Nézzük meg, hogy a c értéke nagyobb-e mint a d?
- Növeljük meg a c értékét 200-al!
- Majd újból nézzük meg, hogy a c értéke nagyobb-e a d-nél?



(06b.py)
Hozzunk létre egy új python fájlt, a neve legyen 06b.py!

Készítsünk programot, melyben True vagy False értéket adunk a,b,c,d,e,f változóknak!
Majd képen látható módon nézzük meg milyen értéket kapnak, ha futtatjuk a kódunkat!



07. GRAFIKUS MEGJELENÍTÉS

Általában a programozási ismereteket az úgynevezett „teknős” útjának rajzolásával szokták kezdeni. Mert látványosabb mint karakteres felületen angol nyelven programozni.
A Python programban is van grafikus felület, ahol sok olyan dolgot ki tudunk próbálni, ami kell az alapok megismeréséhez.

Ahhoz, hogy grafikus módban dolgozzunk, szükség van a program elején begépelni a „**from turtle import ***” parancsot! (Mint a „math” és a „random” függvénykönyvtárnál.)
Az első utasítások melyekkel meg kell ismerkedni:

- **előre menni/haladni: forward(képpont_száma)**
- **fordulni jobbra: right(fok)**
- **fordulni balra: left(fok)**
- **képernyőt törölni: reset()**
- **rajzoló szín megadása: color(„alap_szín_angolul”)**
- **kitöltő szín megadása: fillcolor(„alap_szín_angolul”)**
- **a kitöltendő alakzat előtt és után: begin_fill() … end_fill()**
- **a rajzoló „ceruza” felemelésére az up(), letétele a down() utasítást használjuk**

(07a.py)
Hozzunk létre egy új python fájlt, a neve legyen 07a.py!

Az első grafikus feladatunkban, rajzoljunk egy téglalapot!

- Az első sorban hívjuk meg a rajzoláshoz szükséges eszközöket!
- Menjünk előre 300 képpontot!
- Forduljunk el balra 90 fokot!
- Majd menjünk előre 100 képpontot!
- Aztán gépeljünk be utasításokat, hogy visszaérjünk a kiindulópontba!
- Futtassuk le a programunkat!

07a.py

```
1  from turtle import *
2  forward(300)
3  left(90)
4  forward(100)
5  left(90)
6  forward(300)
7  left(90)
8  forward(100)
```

(07b.py)
Az előző programunkat mentsük el másként 07b.py néven!

Az előző programunkat bővítsük ki, hogy egy zöld színnel rajzolt, sárga kitöltésű téglalapot kapjunk!

- Maradjon az első sorban meghívott rajzoláshoz szükséges eszköztár!
- Töröljük a képernyőt!
- Adjuk meg a zöld rajzolószínt!
- A kitöltőszint állítsuk be sárgára!
- Ahhoz, hogy a kitöltés működjön, meg kell adni az alakzat rajzolásának kezdetekor a begin_fill() utasítást!
- Majd a megrajzolt alakzat végén a lezáró end_fill() utasítást!
- Futtassuk le a programunkat!
- Javítsuk az esetleges hibákat!

07b.py

```
1  from turtle import *
2  reset()
3  color("green")
4  fillcolor("yellow")
5  begin_fill()
6  forward(300)
7  left(90)
8  forward(100)
9  left(90)
10 forward(300)
11 left(90)
12 forward(100)
13 end_fill()
```

Bővítsük az utasítások ismeretét!

(07c.py)

Hozzunk létre egy új python fájlt, a neve legyen 07c.py!

Ebben a programban csak egy egyszerű téglalapot rajzolunk ki, de közben megtanulunk pár hasznos utasítást!

- A program elején **meghívjuk a grafikus módot**! Az első sor megmondja a Pythonnak, hogy töltse be a **turle nevű modult**.
- Az előző modul két új típust hoz be a látótérbe, amelyeket ezután használhatunk: a **Turtle**, azaz **teknőc típust** és a **Screen**, azaz **képernyő típust**. A turtle.Turtle szövegben a pont jelölés azt jelenti, hogy „a Turtle típus, ami a turtle modulban van definiálva”. (Megjegyzés: a Python érzékeny a kis és nagy betűkre, így a modul neve t-vel írva különbözik a Turtle típus nevétől.)
- A 3. sorban létrehozunk, nevet adunk az ablaknak, amiben rajzolunk!
- A 4. sorban nevet adunk annak a „teknőcnek” amivel rajzolunk.
- Majd utasításokkal irányítjuk a teknőcünket. Rajzolunk egy téglalapot.
- A 15. sorban olyan utasítást adunk ki, amely vár a felhasználóra, hogy bezárja az ablakot!
- Tehát az utolsó sor is fontos szerepet játszik: az ablak változó hivatkozik az ablakra, ahogy fentebb bemutattuk. Ha meghívjuk a **mainloop** nevű **metódus**át, belép egy állapotba, ahol egy **eseményre vár** (mint például a billentyűleütés vagy egérmozgatás és kattintás).

07c.py

```
1  import turtle                    # Lehetővé teszi a teknőc használatát
2
3  window = turtle.Screen()         # Hozzunk létre egy "játszóteret" a teknőcnek!
4  teki = turtle.Turtle()           # Hozzunk létre egy teknőcöt "teki" néven!
5
6  teki.forward(100)                # A teki menjen 100 egységet előre!
7  teki.left(90)                    # A teki forduljon 90 fokot!
8  teki.forward(50)
9  teki.left(90)
10 teki.forward(100)
11 teki.left(90)
12 teki.forward(50)
13 teki.left(90)
14
15 window.mainloop()                # Várj, amíg a felhasználó bezárja az ablakot!
```

(07d.py)

Hozzunk létre egy új python fájlt, a neve legyen 07d.py!

Ebben a programban rajzolunk kék színnel, és vastagabb „ceruzával” egy háromszöget!

Parancsok:

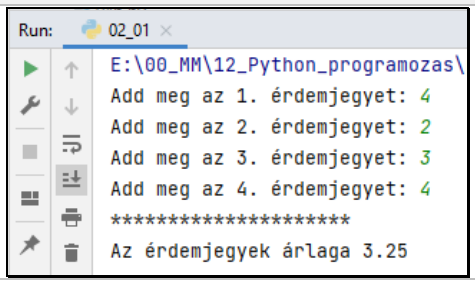
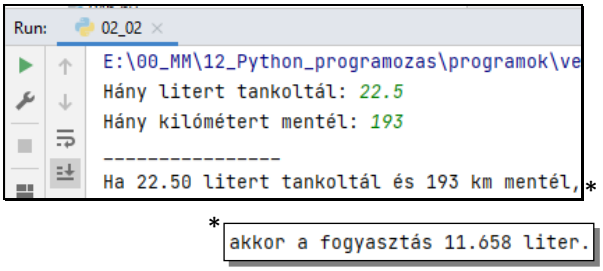
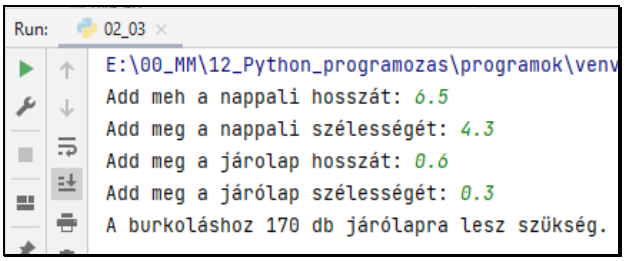
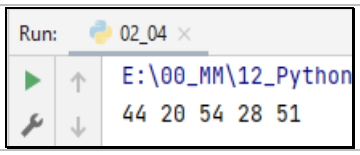
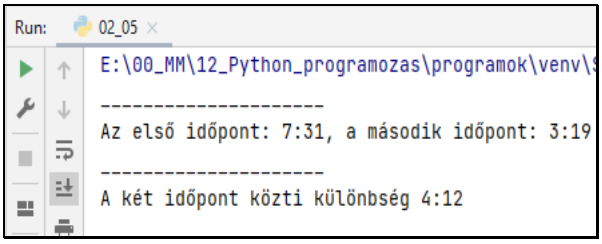
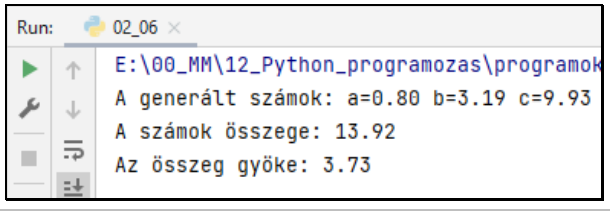
- color(„red”)
- pensize(3)

07d.py

```
1  import turtle
2
3  ablak = turtle.Screen()
4
5  teknoc = turtle.Turtle()
6  teknoc.color("red")          # Mond meg a teknoc-nek, hogy változtasson színt!
7  teknoc.pensize(3)            # Mond meg a teknoc-nek, hogy változtassa meg a tolla vastagságát!
8
9  teknoc.forward(200)
10 teknoc.left(120)
11 teknoc.forward(200)
12 teknoc.left(120)
13 teknoc.forward(200)
14
15 ablak.mainloop()
```

GYAKORLÓ FELADATOK (04-07. témakör)

Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

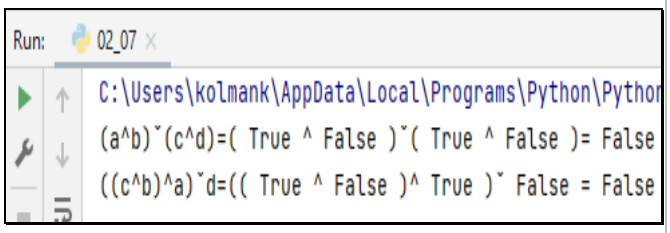
FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (02_01.py) Készíts programot, amely bekér 4 érdemjegyet, majd írasd ki az átlagukat, század pontossággal!	
2. (02_02.py) Készíts programot, mellyel kiszámolod, hogy egy autó mennyit fogyaszt 100 kilométeren! Kérd be, hogy hány litert tankoltál, és hogy hány kilométert mentél! (liter ->float, km->int) Képlet: Fogyasztás=tankolt_mennyiség*100/megtett_km Az eredményt ezred pontossággal írasd ki!	
3. (02_03.py) Burkolni szeretnéd járólappal a nappalidat. Számold ki, hogy darab járólapra lesz szükséged! Készíts programot mely bekéri a nappal egyik, másik oldalát méterben. Nem egész számokat is használhassunk a programban! Majd kérd be egy járólap méretét! Hosszát és a szélességét! (A járólapok mérete kisebb, mint 1 méter. pl.: 0,3m*0,3m, 0,3m*0,6m, 0,3m*0,2m) Arra figyelj, hogy mindig 10%-al túl kell tervezni a szükséges mennyiséget! A végső darabszám viszont egész legyen!	
4. (02_04.py) Készíts olyan programot, mellyel ötöslottó sorsolást modellezel! (Az ötöslottóban 1-90-ig vannak számok.)	
5. (02_05.py) Készíts egy programot, melyben generálsz két véletlenszerű időpontot órában és percben meghatározva, majd jelenítsed meg a két időpont közti különbséget! Használd az abs() függvényt!	
6. (02_06.py) Generálj három darab véletlen valós számot 1 és 10 között, majd szorozd össze őket, és végül vonjál gyököt a szorzatból! Készíts programot, hogy a képernyőkép úgy nézzen ki, ahogy a mintán!	

7. (02_07.py)

Készíts programot, melyben az elején megadsz négy változóban (a,b,c,d) True és False értékeket, majd a következő képleteket old meg, a képernyőkép alapján! (ˇ➔vagy, ^➔és)

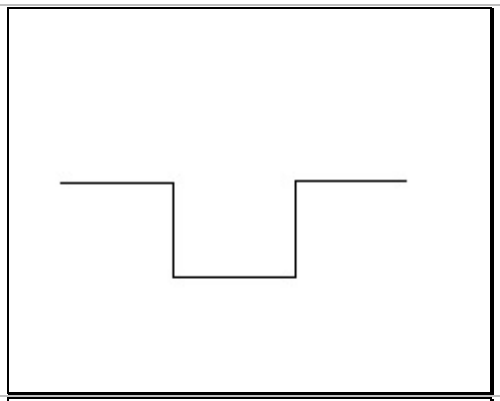
(a^b)ˇ(c^d)

((c^b)^a)ˇd



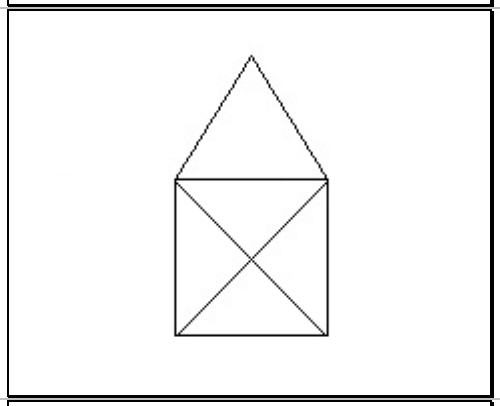
8. (02_08.py)

Rajzold meg a jobb oldalon lévő ábrát!
A szakaszok egyforma hosszúságúak, 100 kp!
A program lefutás után várjon, amíg a felhasználó bezárja az ablakot!



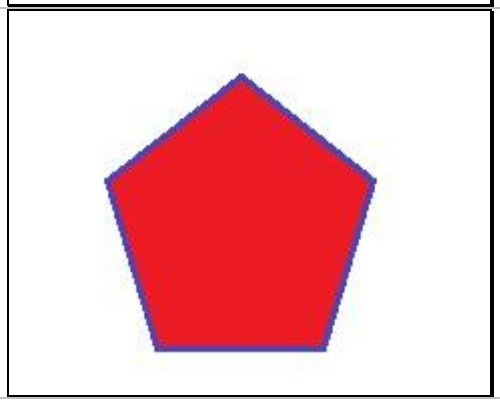
9. (02_09.py)

Rajzold meg a jobb oldalon lévő „ház” ábrát úgy, hogy egyetlen vonalat használsz! (magyarul nem emelheted fel a „tollat”!
A „ház” oldalai 100 képpontosak, viszont van olyan vonal, aminek a méretét Pithagorasz tétellel ki kel számolnod!
A program lefutás után várjon, amíg a felhasználó bezárja az ablakot!



10. (02_10.py)

Rajzolj kék színnel egy piros kitöltéses ötszöget!
Az oldalak hossza legyen 100 képpont!
A program lefutás után várjon, amíg a felhasználó bezárja az ablakot,
vagy használjuk a time.sleep() parancsot és várjon az ablak/program bezárásával 2 másodpercet!



Hasznos parancs a grafikus felület használatakor a time.sleep():
Ha a rajzolt alakzatnál azt szeretnénk, hogy ne azonnal ugorjon ki a programból a lefutás végén, akkor a kódunk végére írjuk be a **time.sleep(2.5)** utasítást! Ami azt eredményezi, amikor megrajzolta az eredeti utasítások alapján az alakzatot, akkor 2,5 másodpercet vár. A zárójelben mindig float számot kell megadni, másodpercben!
A kód elején meg kell adni az „**import time**” függvény könyvtárt, hogy tudjuk használni a time.sleep() parancsot! (Olyan mint a turtle és a math.)
A PyCharm lehet, hogy összevonja a program elején az import utasításokat.

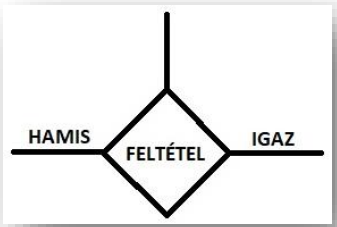
```
1   import time
2   from turtle import *
```

```
6   time.sleep(2.0)
```

```
1   import ...
1   import time
2   from turtle import *
```

08. FELTÉTELES ELÁGAZÁS -IF

Programozásban fontos az utasítások sorrendje. Ha ugyanazon utasítások sorrendjét felcseréljük, akkor más lesz a végkifejlet. A **vezérlési szerkezet**ek alkalmazásával dönthetjük el, hogy melyik utasítás következzen.
Az eddigi programjainkban **sorrendi végrehajtást (szekvenciát)** alkalmaztunk.
Mostantól pedig **elágazásokat (szelekciókat)** fogunk használni.
Ha egy feltételtől függően a program többféleképpen folytatódhat, akkor **elágazást alkalmazunk**.
A feltétel egy logikai érték, tehát lehet igaz (True), vagy hamis (False). Attól függően, hogy egy vizsgálatra adott válasz igaz, vagy hamis, különböző irányban folytatódik a program.



Öt fajta elágazást fogunk megnézni:

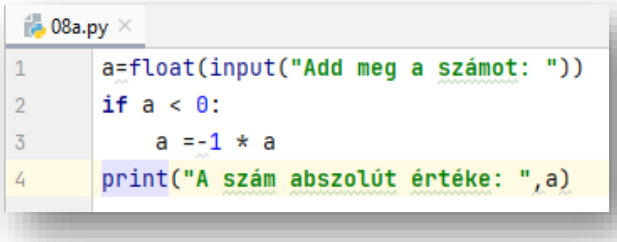
- **Elágazás egy irányban (if)**
- **Elágazás két irányban (if-else)**
- **Többszörös elágazás (if-elif-else)**
- **Egymásba ágyazott elágazás (if-else-if-else)**
- **Elágazás összetett feltétellel (not, and, or)**

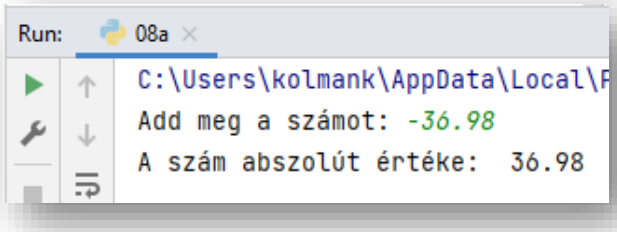
Elágazás egy irányban (if)

Egyirányú elágazásról akkor beszélünk, ha csak az igaz ágat hozzuk létre!
Mondatszerű leírás: **Ha <feltétel> akkor <utasítások>**
Elágazás vége
Az előző mondat értelmezése a következő: a feltétel teljesülése esetén az „akkor mögötti utasítások végrehajtódnak. Ha a feltétel nem teljesül, akkor ezt a feladatrész átugorva folytatódik a program végrehajtása.

(08a.py)
Hozzunk létre egy új python fájlt, a neve legyen 08a.py!

A feladatban számítsuk ki egy bekért szám abszolút értékét. Írjuk meg az algoritmust úgy, hogy ha a szám nem negatív, akkor semmit nem kell tenni a számmal, ha viszont negatív, akkor a számnak kell venni a mínusz egyszeresét. (Most nem az abs() függvényt használjuk.)





A nagyon rövid program **mondatszerű leírás**a a következő:

(1) Be: a
(2) Ha a<0 akkor
(3) a = -1*a
(4) Ki: a

- A feladatot úgy oldjuk meg, hogy először bekérünk egy valós számot, ami lehet pozitív, vagy negatív.
- Majd indítunk egy **„if" feltételt**, ahol megvizsgáljuk, hogy az „a" változó kisebb-e mint nulla. Mert, ha igen a válasz a feltételre, akkor végrehajtóik a következő (automatikusan beljebb kezdődő) következő sor.
- A feltétel után a Pythonban : (kettőspon) karaktert kell elhelyezni. **Minden vezérlési szerkezetnél meg kell adni a :-ot.** Tehát az if elágazás miatt kell a „:"-ot beírni!
- Ha a válasz nem, akkor átugorja a beljebb kezdődő utasításoka, és csak egyszerűen kiírja az eredeti számot!

Elágazás két irányban (if-else)

A kétirányú elágazásnál, a feltételtől függően mind a két irányba különböző utasítások hajtódnak végre! Tehát a hamis ágban is lesznek utasítások.
Mondatszerű leírása a következő:

Ha <feltétel> akkor <utasítások1>
különben <utasítások2>
Elágazás vége

Az előzőekben leírtak értelmezése szerint , ha a megadott feltételre igen a válasz, akkor az „utasítások1” hajtódik végre, ha pedig nem a válasz a feltételre, akkor az <utasítások2> hajtódik végre! Ezután az „Elágazás vége” után folyatódik a program.

(08b.py)
Hozzunk létre egy új python fájlt, a neve legyen 08b.py!

Számítsuk ki egy bekért szám négyzetgyökét. Ha a szám negatív, akkor írassuk ki, hogy a feladat nem végrehajtható!

- A programban az első sorban elérhetővé tesszük az sqrt() függvényt a „from math import *” utasítással!
- Bekérjük a valós számot!
- Létrehozunk egy elágazást if utasítással, az utána következő feltétellel és a kettősponttal!
- Az „igaz” ágban kiíratjuk a szöveget! Fontos, hogy a program automatikusan beljebb kezdi az ág utasításait. Ezen nem szabad változtatni!
- Ha véletlenül nem kezdődik beljebb, akkor nekünk kell Tabot nyomi, vagy négy szóközt!
- Aztán jön egy „else” (különben) ág, melyben a hamis válasz után történő utasításokat helyezzük. Jelen esetben a gyökvonást!
- Futtassuk a programot, mind a két eshetőséget teszteljük. Azesetleges hibákat javítsuk!

```
from math import *
a=float(input("Add meg a számot: "))
if a < 0:
    print("Nem végezhető el a művelet!")
else:
    print("A szám gyöke: %.2f" % (sqrt(a)))
```

```
Run: 08b
C:\Users\kolmank\AppData\Loca
Add meg a számot: -35.14
Nem végezhető el a művelet!
```

```
Run: 08b
C:\Users\kolmank\AppData\Lo
Add meg a számot: 36.47
A szám gyöke: 6.04
```

A Pythoon programozási nyelvben a következő módon használjuk az összehasonlító relációs jeleket!

Összehasonlítás	Mondatszerű leírás Algoritmusok	Python
nagyobb	a>b	**a>b**
kisebb	a<b	**a<b**
nagyobb egyenlő	a>=b	**a>=b**
kisebb egyenlő	a<=b	**a<=b**
egyenlő	a=b	**a==b**
nem egyenlő	a<>b	**a!=b**

Ezeket nagyon meg kell jegyezni, mert sokszor fogjuk használni!

Elágazás több irányban (if-elif-else)

Ha kettőnél több kimenet van, akkor alkalmazhatunk „elif” ágakat is. Tehát több feltételt vizsgálhatunk!
Nagyon fontos viszont a feltételek sorrendje, mert **fentről lefelé indulva vizsgálja a feltételeket**, és **az első igaz állításra kilép**, és az **elágazás után folytatja**! Az első if feltétel után elif feltételekkel adhatjuk meg az összehasonlítás lehetőségeit, majd az utolsó ág itt is a „különben” else ág!

(08c.py)
Hozzunk létre egy új python fájlt, a neve legyen 08c.py!

Ebben a feladatban kérjünk be egy számot, melyről állapítsuk meg, hogy pozitív, nulla, vagy negatív!

- Bekérünk egy valós számot!
- Az első (if) vizsgálatnál megnézzük, hogy a szám nagyobb-e mint 0! Ha igen, akkor végrehajtódik az első ptint() utasítás.
- A második vizsgálat már az „elif” utasítás mögött van, ahol azt vizsgáljuk, hogy egyenlő-e nullával! Mert ha igen, akkor végrehajtódik a második print() utasítás.
- A különben ágban pedig már csak a negatív számok maradtak, nem kell vizsgálni semmit, csak a szöveget kiíratni!
- Teszteljük mind a három lehetőségre!

```
08c.py
1  a=float(input("Addj meg egy számot: "))
2  if a>0:
3      print("A szám pozitív.")
4  elif a==0:
5      print("Nulla.")
6  else:
7      print("A szám negatív")

Run:  08c
C:\Users\kolmank\AppData\Loca
Addj meg egy számot: -26.13
A szám negatív
```

Egymásba ágyazott feltételvizsgálat (if-else-if-else)

Az előző feladatot megadhatjuk másféleképpen is. Nem „elif” utasítás használatával, hanem az if-else egymásba ágyazásával.

(08d.py)
Hozzunk létre egy új python fájlt, a neve legyen 08d.py!

A feladat ugyan az mint az előbb!

- Ebben az esetben is bekérünk egy valós számot!
- Csak if és else ágakat hozunk létre, a különböző lehetőségek végig vételével!
- Fontos, hogy amikor új fi ágat indítunk az utasításokat beljebb kell kezdeni!
- A minta programon látszik a tagolás. Ezt minden esetben be kell tartani!
- Teszteljük mind a három lehetőséget!

```
08d.py
1  a=float(input("Add meg a számot: "))
2  if a>0:
3      print("Pozitív.")
4  else:
5      if a==0:
6          print("Nulla.")
7      else:
8          print("Negatív")

Run:  08d
C:\Users\kolmank\AppD
Add meg a számot: 0
Nulla.
```

Elágazás összetett feltétellel (not, and, or)

Az összetett feltételek részfeltételekből állnak, amelyeket a logikai műveletekkel tudunk egymás után helyezni. Az if feltétel után írjuk be a megfelelő képletet.
Ilyen feladat lehet például, hogy a változó értéke az 50 nél nagyobb, de 100-nál kisebb szám-e.
Logikai műveletek segítségével tudunk több logikai értékből egyet készíteni, összetett feltétellé összekapcsolni. Tehát az előző példát megnézve: (a>50 and (a<100)

A logikai műveleteket átismételve (True, False):

- **AND ➔ Akkor igaz, ha minden állítás igaz!**
- **OR ➔ Akkor igaz, ha legalább az az egyik állítás igaz!**
- **NOT ➔ Az ellenkezőjére változtatja az értéket!**

(08e.py)
Hozzunk létre egy új python fájlt, a neve legyen 08e.py!

Egy angol nyelvvizsgán az írásbelin 100, szóbelin 50 pontot lehet elérni. A vizsgán csak akkor mehetsz át, ha mind a két vizsgán elérted a minimum 60%-ot!
Készítsünk programot, amely bekér két pontszámot, majd eldönti, hogy átmentél e a vizsgán.

```
i=int(input("Add meg az írásbeli eredményét (0-100): "))
sz=int(input("Add meg a szóbeli eredményét (0-50): "))
iszaz=i/100
szszaz=sz/50
#print(iszaz,szszaz)
if (iszaz>0.6) and (szszaz>0.6):
    print("Sikeres a vizsga.")
else:
    print("Sikertelen a vizsga.")
```

```
Run: 08e
C:\Users\kolmank\AppData\Local\Programs\Pyt
Add meg az írásbeli eredményét (0-100): 81
Add meg a szóbeli eredményét (0-50): 44
Sikeres a vizsga.
```

- Először bekérjük az elért pontokat.
- Aztán kiszámoljuk a százalékokat, két változóba!
- Majd kezdjük a HA függvényt, megadjuk „és” feltétellel a vizsgálatokat, sikeres vizsga esetén.
- Az „else” ágban pedig a sikertelen vizsga esetén való kiíratást adjuk meg.

(08f.py)
Hozzunk létre egy új python fájlt, a neve legyen 08f.py!

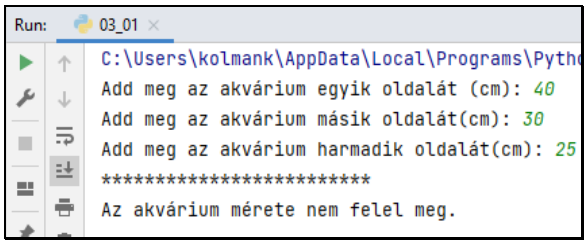
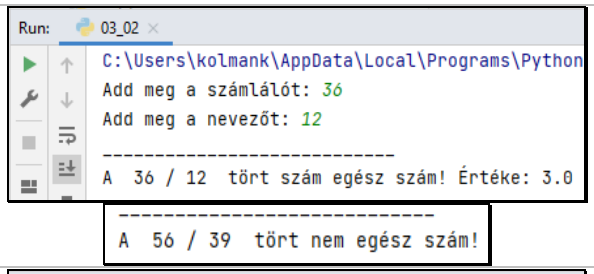
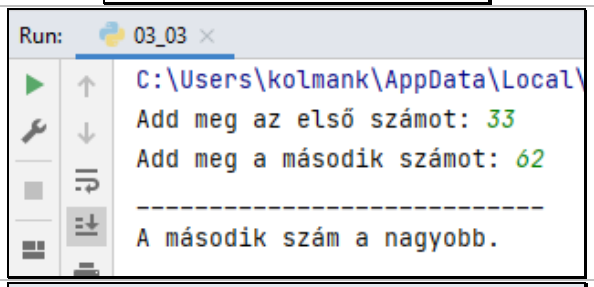
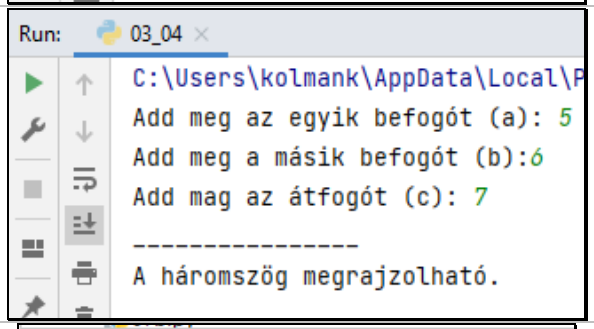
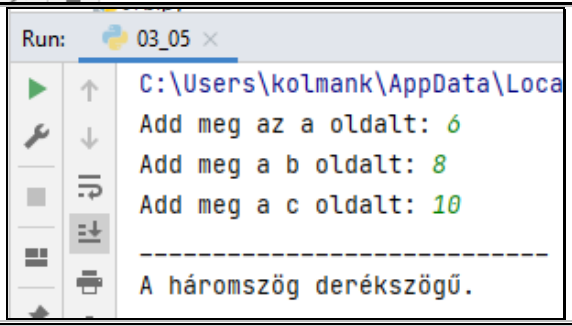
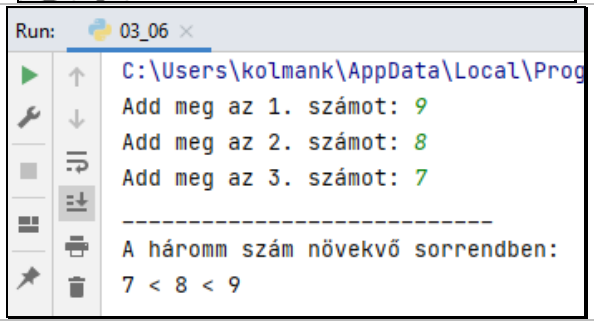
Egy matematika dolgozat, négy nehéz feladatból áll. Ha bármelyik feladatot sikerült megoldani 75%-ra, akkor megkapja a legjobb jegyet! Minden feladat 20 pontos! Készíts programot a képernyőkép alapján! (A végén: „A dolgozat sikeres.” vagy a „A dolgozat sikertelen.” szöveg jelenjen meg!

```
Run: 08f
C:\Users\kolmank\AppData\Local\
Add meg az 1. pontszámát: 21
Add meg az 2. pontszámát: 35
Add meg az 3. pontszámát: 41
Add meg az 4. pontszámát: 49
A dolgozat sikeres.
```

Hasznos parancsok: **break, exit, quit**
Ha bármikor azt szeretnénk, hogy a program kilépjen az aktuális cikusból akkor, egyszerűen, minden paraméter nélkül használjuk a **break** parancsot!
Ha a programból is ki a karunk lépni, akkor használjuk az **exit()** vagy a **quit()** parancsokat.

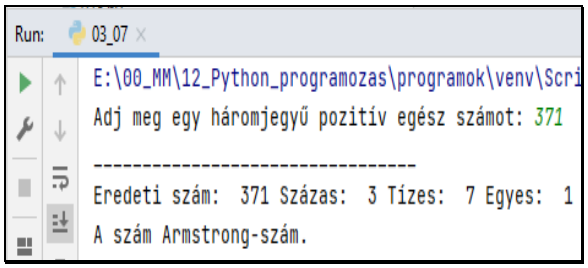
GYAKORLÓ FELADATOK (08. témakör)

Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (03_01.py) Készíts programot, amelyben egy akváriumot készítünk! A halaknak 50000 cm3 vízre van szükség. A program kérjen be három egész számot, amelyek az oldalak hosszúságát jelentik. A kérdés az, hogy megfelelnek-e az adatok?	
2. (03_02.py) Kérjünk be két egész számot, amely egy közönséges tört számlálója és nevezője. Döntsük el, hogy a bevitt tört felírható-e egész számként. Ha igen, akkor írja ki az egész számot, különben írja ki, hogy nem egész!	
3. (03_03.py) Készíts programot, amely bekér két egész számot, majd kiírja, hogy az első szám kisebb, nagyobb, mint a második, vagy egyenlő!	
4. (03_04.py) Készíts programot, mely bekéri egy háromszög két befogóját (a,b) és átfogóját (c), majd kiírja, hogy meg lehet-e rajzolni a háromszöget! A feladat megoldásához tudni kell, hogy mi a szabály!	
5. (03_05.py) Készíts programot, mely bekéri egy derékszögű háromszög két befogóját (a,b) és átfogóját (c), majd kiírja, hogy a háromszög tényleg derékszögű-e! A feladat megoldásához használd a Pitagorasz - tételt!)	
6. (03_06.py) Készíts programot, amely bekér három pozitív egyész számot és kiírja őket nagyság szerint!	

7. (03_07.py)

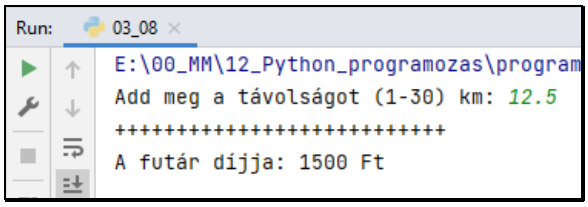
Készítsünk egy programot, amely bekér billentyűzetről egy három jegyű pozitív egész számot, és eldönti róla, hogy Armstrong-szám-e! A háromjegyű Armstrong-számokra igaz, hogy a számjegyei köbének összege megegyezik az eredeti számmal! pl.: $371=3^3+7^3+1^3=27+343+1$



8. (03_08.py)

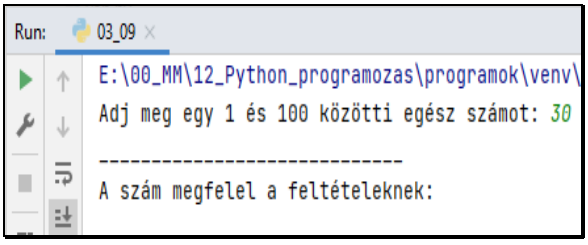
Egy biciklis futár az egyes utakra az út hosszától függően kap fizetést (1-30 km)! A lenti táblázat alapján készíts programot, amely kiírja, hogy hány forintot kap! (Bevitel lehet nem egész szám.)

1-5 km	500 Ft
6-10 km	1000 Ft
11-15 km	1500 Ft
16-20 km	2000 Ft



9. (03_09.py)

Olyan számokat keresek 1 és 100 között, amelyek párosak, és oszthatók hárommal és öttel is!
Készíts programot, amely bekér egy egész számot majd eldönti, hogy megfelel-e a fenti követelményeknek!
Aztán kiírja, hogy: „A szám megfelel a feltételeknek.” vagy „A szám nem felel meg a feltételeknek.”

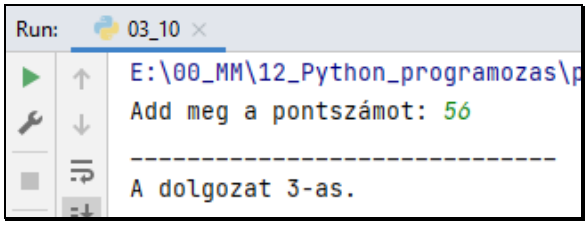


10. (03_10.py)

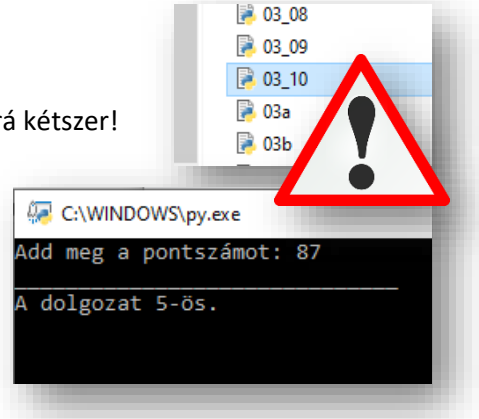
Egy digitális kultúra dolgozat 67 pontos összesen. Készíts programot, ami kiírja, hogy hányas a dolgozat, ha a következő százalékok tartoznak a különböző érdemjegyekhez!

0-40%➔1
41-55%➔2
56-70%➔3
71-85%➔4
86-100%➔5

A pontok egész számok legyenek!



Az elkészített programok futtatása nem PyCharm környezetben:



- A futtatandó programot keressük meg az intézőben, és kattintsunk rá kétszer!
- A Windows egy külön ablakban elindítja a kódunkat.
- A karakteres programot fekete képernyőn, fehér karakterekkel fut.
- Grafikus módban fehér felületen, fekete rajzolószínnel fut.
- Azért, hogy a lefutás végén ne azonnal ugorjon ki, alkalmazzunk néhány másodperc késleltetést a program végén a time.sleep(2.0) utasítással!

09. FOR CIKLUSOK (FOR –RANGE)

A programozásban nagyon sokszor előfordul, hogy egy műveletet, vagy műveletsort ismételni kell. A programrészek ismétlését **ciklusok** segítségével oldjuk meg. A ciklusoknak több fajtája van. Mi a legalapvetőbbeket nézzük meg. Ezek a **számláló és a feltételes ciklus.**

Számláló ciklus

Ha a program írásakor el tudjuk dönteni, hogy hányszor szükséges lefutnia a programrészletnek, akkor számláló ciklust alkalmazunk. Például, ha egy karaktert, tízszer ki szeretnék íratni a képernyőre, akkor a számláló ciklust alkalmazunk.

- A megvalósításhoz szükségünk lesz egy úgynevezett **ciklusváltozó**ra.
- Ennek a típusának **megszámlálható**nak kell lennie!
- Programozói szokás, ezeket **a változókat i-nek és j-nek hívn**i.
- Ennek a változónak kell adni egy **kiinduló érték**et.
- A ciklus addig ismétel, amíg el nem éri a **végérték**et.
- Van egy úgynevezett **növekmény** is, amely meghatározza, hogy mekkora léptekkel halad a ciklus.
- Nagyon fontos a ciklus tagolása (beljebb kezéés).

A ciklus **mondatszerű leírás**a a következő:

ciklus i:=1-tól 10-ig 1-esével
Ki: „X”
ciklus vége
Ki: „Ciklus utáni utasítás.”

i ➔a ciklusváltozó
1➔a ciklusváltozó
10➔végérték
1-esével➔növekmény

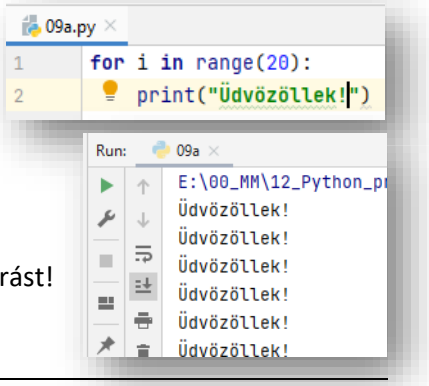
A Python programozáskor a „**for i in range(0,10,1)**” utasításra van szükségünk a ciklus megadásához.

- A for utasítás után megadjuk a ciklusváltozó nevét, majd az in szócska után a range utasítás zárójeles részében először a kezdő, majd a végértéket, végül a növekményt adjuk meg.
- Ha a növekmény 1, akkor elhagyhatjuk azt, elég csak az első két számot megadni!
- Ha a kezdőérték 0, akkor azt sem kell megadni, így csak a végértéket írjuk be a zárójelbe.
- Ha a range utasításnak egy értéke van, az mindig a végértéknél egyel nagyobb szám. Tehát ha valamit szeretnénk 10-szer lefuttatni, akkor 11-et aduk meg.
- A ciklus paramétereinek megadása után kettőspontot kell tenni, mert vezérlési szerkezet.
- A ciklus végét külön nem jelezzük, hanem a behúzást szüntetjük meg.

Megjegyzés	Mondatszerű leírás	Python
Egyparaméteres, kezdőérték: 0 végérték: 100 növekmény: 1	ciklus i:=0-től 100-ig <ciklusmag> ciklus vége	for i in range(101) <ciklusmag>
kétparaméteres, kezdőérték: 0 végérték: 100 növekmény: 1	ciklus i:=0-től 100-ig <ciklusmag> ciklus vége	for i in range(0,101) <ciklusmag>
háromparaméteres kezdőérték: 0 végérték: 100 növekmény: 10	ciklus i:=0-től 100-ig 10-esével <ciklusmag> ciklus vége	for i in range(0,101,10) <ciklusmag>

(09a.py)

A legegyszerűbb módja a ciklus kipróbálásának az az, hogy írassunk ki a képernyőre valamilyen szöveget n-szer!
Ebben a konkrét példában az „Üdvözöllek!” szöveget írassuk ki 20-szor!

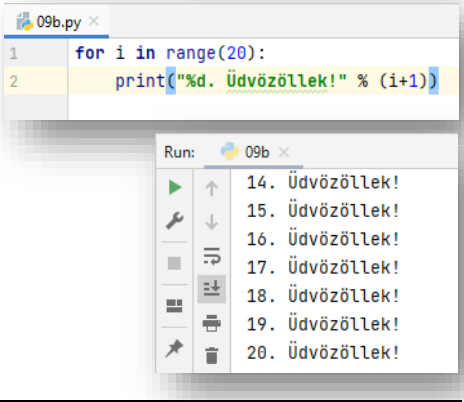


- A for parancs után megadjuk az „i” –t, mint változót, ami 0 –ról indul.
- A range(20) , az első sor végén megadja, hogy 20-szor hajtsa végre a kiírást!
- A print utasításban megadjuk a kiírandó szöveget!

(09b.py)

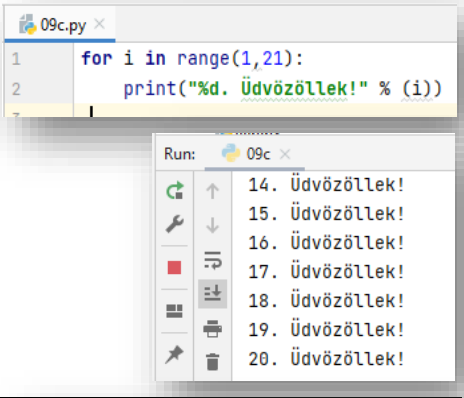
Módosítsuk az előző programot úgy, hogy a sor elején jelenjen meg, hogy éppen hányadszorra írta ki! Hányadik sorban vagyunk!

A szokott módon adjuk meg a formátumot, csak arra kell figyelni, hogy az „i” értéket egyel növelni kell!



(09c.py)

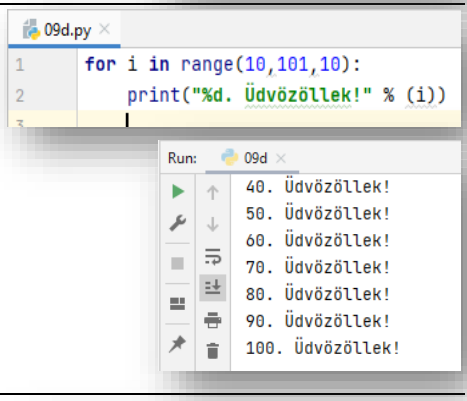
Ha a ciklusban kezdőértéket állítunk be, akkor nem kell egyesével növelni az „i”-t. Ilyenkor a ciklus kétparaméteres alakját használjuk. Mivel most 1-től szeretnének 20-ig kiíratni a számokat, ezért a végétéket 21-re kell állítani.



(09d.py)

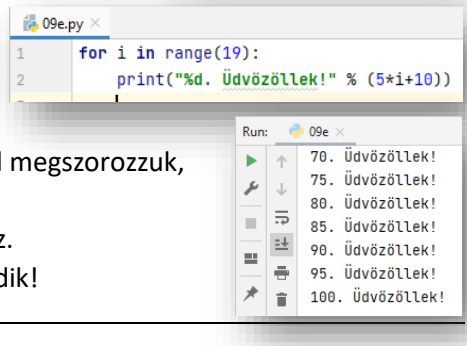
Módosítsuk a programot úgy, hogy a program 10-től 100-ig tízesével írja ki a sorszámokat!

Mivel itt a növekményt is megadjuk, ezért háromparaméteres formában írjuk be a range parancsba, az adatokat.



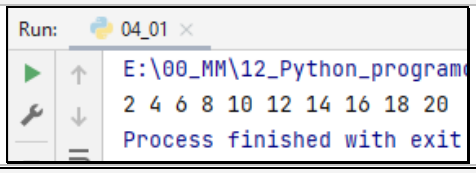
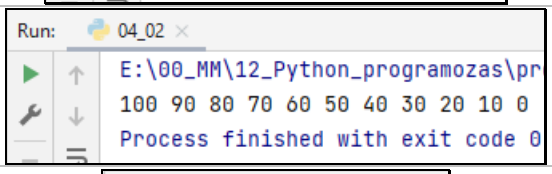
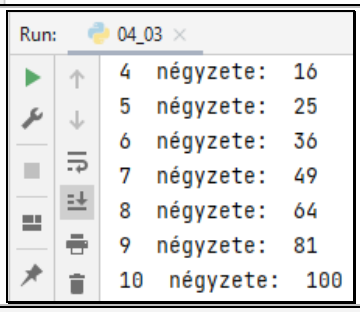
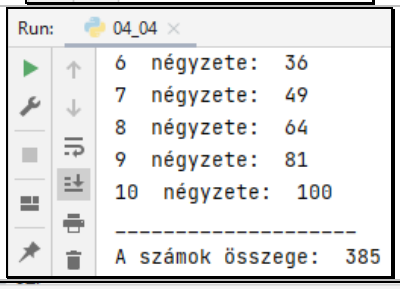
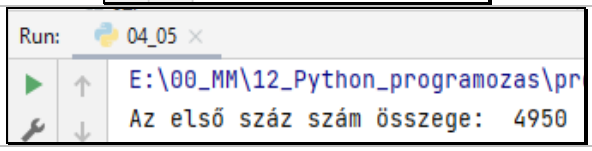
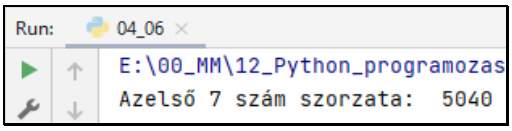
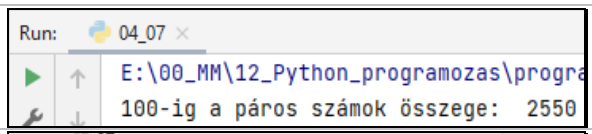
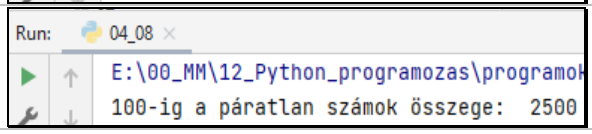
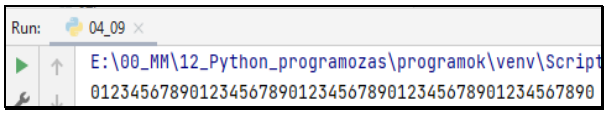
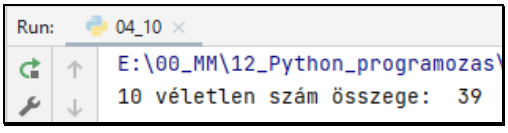
(09e.py)

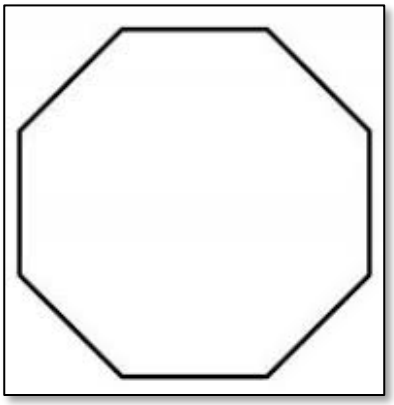
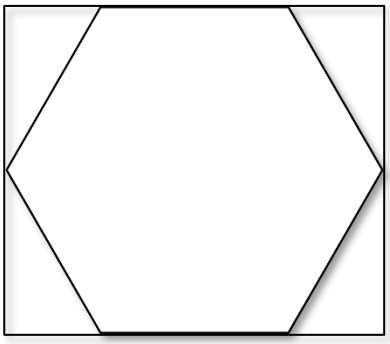
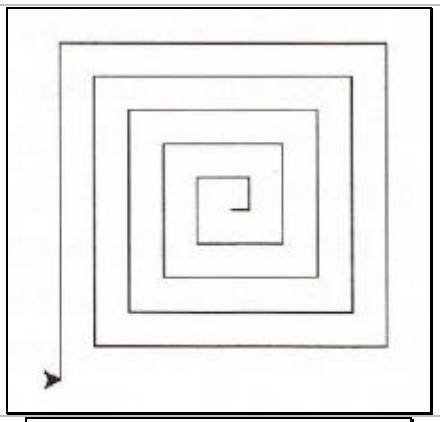
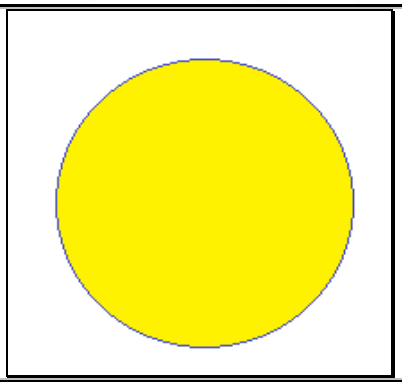
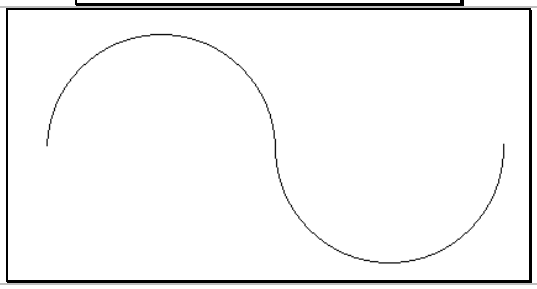
Végül nézzünk egy olyan példát, amit egyparaméteresen oldunk meg.
Írjuk ki 10-től ötösével 100-ig a sorszámokat!
Ha az „i” értékét eredetileg 1-esével emeljük, de bent a ciklusban 5-el megszorozzuk, akkor 5-ösével fog lépkedni és azt is írja ki.
Ha pedig hozzá adunk mindig 10-et, akkor a kiinduló szám az a 10-lesz.
Nézzük meg a példaprogramban, és értelmezzük, hogy hogyan működik!



GYAKORLÓ FELADATOK (09. témakör)

Készítsd el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (04_01.py) Írasd ki az első 10 páros számot, egymás mellé, szóköznyi távolsággal!	
2. (04_02.py) Írasd ki 100-tól 0-ig 10-esével a számokat visszafelé, szóközökkel elválasztva!	
3. (04:03.py) Készíts programot, amely, a képernyőképen látható módon kiírja 1-10-ig a számok négyzetét!	
4. (04_04.py) Bővítsed ki az előző programot a minta szerint úgy, hogy a végén egy vonal alatt írasd ki a négyzetszámok összegét! A feladat megoldásához vezess be egy „ossz” nevű változót. Melyben eltárolod és hozzáadod -futás közben - az aktuális összeget a változóhoz.	
5. (04_05.py) Írasd ki az első száz szám összegét!	
6. (04_06.py) Írasd ki az első 7 szám szorzatát! Itt figyelni kell arra, hogy az a változó, amiben a szorzatot tároljuk, kezdetben nem lehet nulla.	
7. (04_07.py) Számold ki 100-ig a páros számok összegét!	
8. (04_08.py) Számold ki 100-ig a páratlan számok összegét! írd át az előző programot!	
9. (04_09.py) Írasd ki a mintán látható számsort! (A cikusváltozót oszd el 10-el és a maradékot írd ki!	
10. (04_10.py) Készíts programot, melyben generálsz 10 db véletlen számot 1 és 10 között, majd összeadod őket!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
11. (04_11.py) Készíts grafikus felületen, a for ciklus felhasználásával egy nyolcszöget! A program lefutása után várjon a program 3 másodpercet!	
12. (04_12.py) Az előző programot metsd el másként! Változtasd meg úgy, hogy hatszöget rajzoljon ki, és lefutás utám várjon 4 másodpercet!	
13. (04_13.py) Készíts programot melyben a jobb oldalon látható ábrát rajzoltatod ki for ciklussal! A programban 10 képpontal növeld a távolságokat! A lefutás után várjon 4 másodpercet!	
14. (04_14.py) Rajzolj egy kört a minta alapján! A kilépés előtt várjon a program 2 másodpercet! A rajzolószín kék legyen, a kitöltőszín sárga! (360 fokban előre kell menni 1 kp-ot, majd elfordulni 1 fokot balra!)	
15. (04_15.py) Készíts programot, amely kirajzolja a következő görbét! A kirajzolás után 2 másodperccel késleltesd a bezárást! A programban két for utasítást kell egymás után beírni a kódba!	

10. EGYMÁSBA ÁGYAZOTT FOR CIKLUSOK

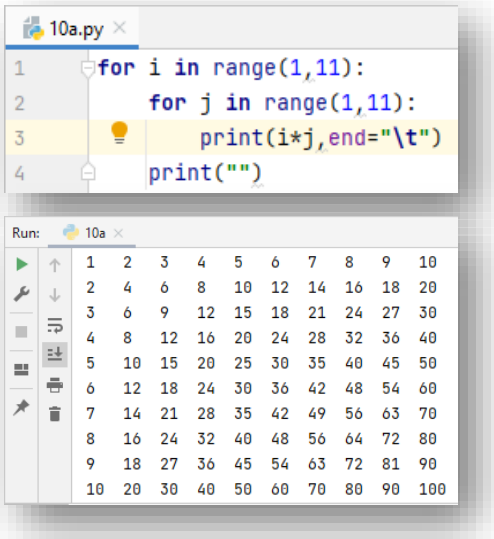
Programjanik készítésekor sokszor elő fog fordulni, hogy nem lesz elég egy ciklust használnunk, mert a **cikluson belül egy újabb ciklust kell indítani** ahhoz, hogy megvalósítsunk a feladatunkat. Ezt hívjuk **egymásba ágyazás**nak. Például egymásba ágyazott ciklust alkalmazunk téglalapszerű (mátrixos) adatelrendezések kiíratásához.

Nézzük példának egy 10*10-es szorzótábla kiíratását. A **mondatszerű leírás** a következő:

ciklus i:= 1-től 10-ig 1-esével
ciklus j:=1-től 10-ig 1-esével
Ki: i*j
Ki: szóköz
ciklus vége
Ki: „új sor”
ciklus vége

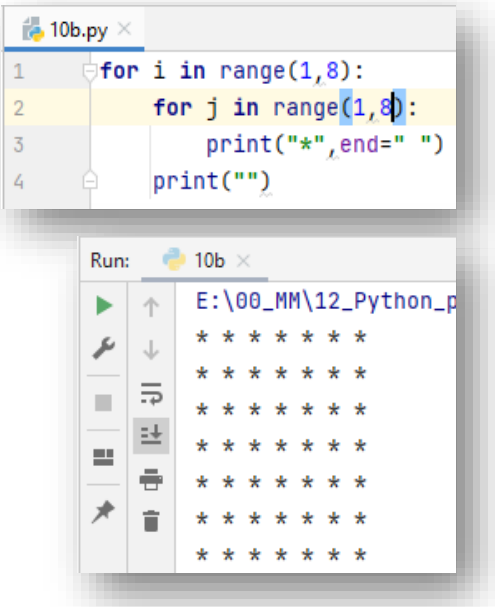
(10a.py)

Valósítsuk meg az előzőekben felvázolt szorzótáblát!
Ezeknél a feladatoknál az „i” és „j” változókat szoktuk használni.
Nézzük a lépéseket:
Az első for ciklus „i”-je 1 értéket kap.
A belső for ciklus „j”-je is 1-es értéket kap.
A harmadik sorban összeszorozza az i-t és a j-t, ami 1, amit kiír.
Aztán **letesz egy tabulátort az end=”\t” paranccsal**. (Nem sortörést.)
Így a kiíratás a minta szerint jobban néz ki!
Majd a belső for-ban halad végig 2-10-ig, szoroz, kiír, szóköz.
Amikor végzett a belső for-al, akkor jön egy sortörés, majd kiugrik külső for-ba. Ott az i értéke növekszik 2-re, majd újból kezdődik a belső ciklus.
Végig halad mátrix szerűen a kiíratással.



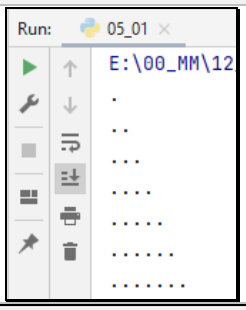
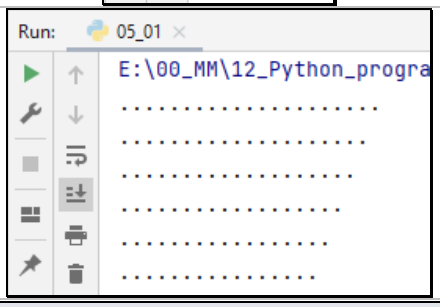
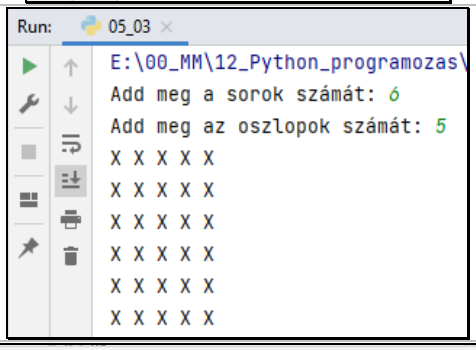
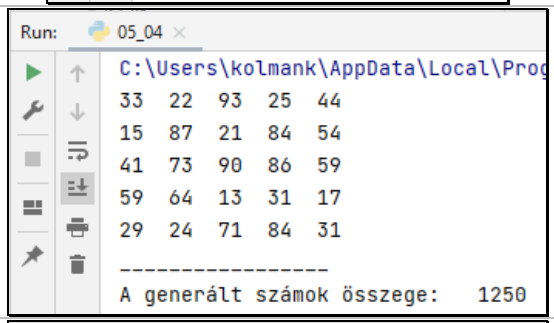
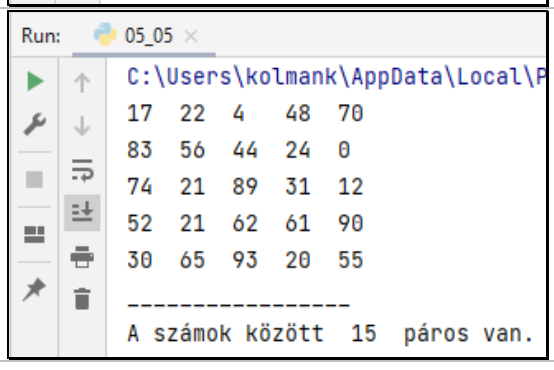
(10b.py)

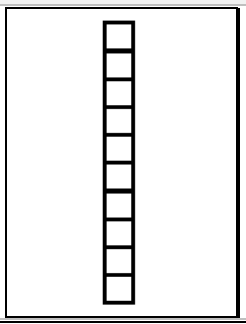
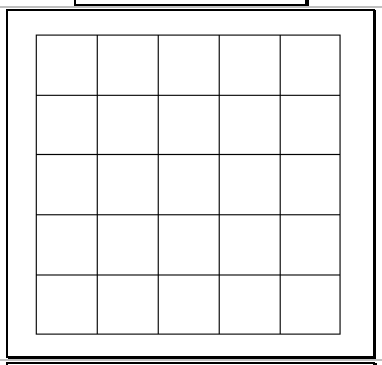
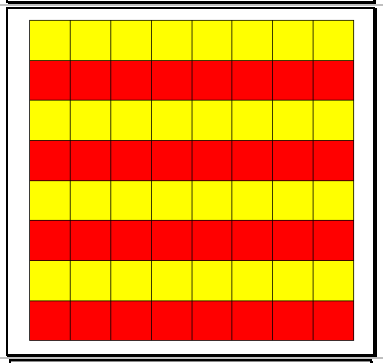
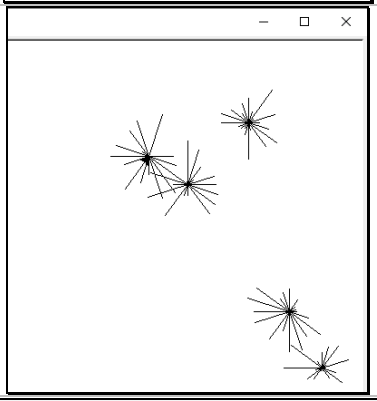
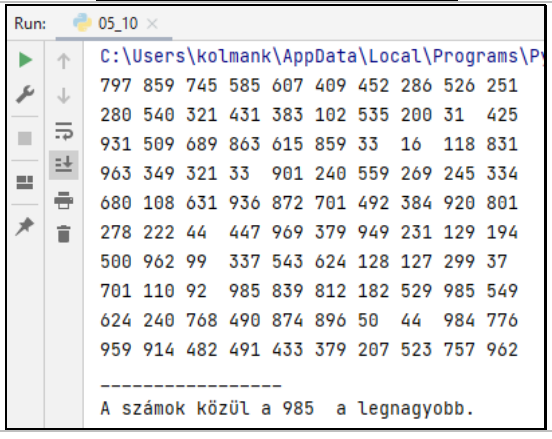
Ebben a feladatban „*” karaktereket fogunk kiíratnia képernyőre.
Állítsuk be, hogy egy 7*7-es mátrixban jelenítse meg a karaktereket!
Mentsd el másként az előző programot, majd változtasd meg a kódot!



GYAKORLÓ FELADATOK (11. témakör)

Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (05_01.py) Készíts programot, melyben „.” karaktereket íratunk ki, egyenként növekvő számban!	
2. (05_02.py) Mentsd el az előző programot másként! Majd írd át a kódot úgy, hogy most csökkenjenek a pontok számai soronként! 20-1-ig.	
3. (05_03.py) Készíts programot, ami bekér két számot, amelyek közül az első az a sorok száma, a második az oszlopok száma. Majd kirajzolja „X” (nagy x) karakterekből a mátrixot, a minta szerint!	
4. (05_04.py) Generálj egy 5*5-ös mátrixba véletlen számokat 0-100-ig! A program végén írasd ki a generált számok összegét a minta alapján! Alkalmazz tabulátort a megjelenítésnél! (end=”\t”)	
5. (05_05.py) Az előző programot mentsd el másként! Változtasd meg a kódot úgy, hogy a generált véletlen számok közül számolja meg a párosokat! Alkalmazz tabulátort a megjelenítésnél! (end=”\t”)	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
6. (05_06.py) Készíts programot, egymásba ágyazott ciklusokkal, melyben a jobb oldalon lévő „létrát” rajzoltatod ki! A kis négyzet oldala legyen 25 képpont! A program lefutása után várjon, amíg a felhasználó bezárja az ablakot!	
7. (05_07.py) Mentsd el az előző programot másként! Majd írd át a kódot úgy, hogy a jobb oldali 5*5-ös táblázatot rajzolja ki a képernyőre! A kis négyzetek oldalai itt is maradjanak 25 képpontosak! A végén 2 másodpercet várjon a program mielőtt kilép!	
8. (05_08.py) Készíts programot, amely a mintaszerinti 8*8-as táblázatot elkészíti! A sorok felváltva legyenek sárgák és pirosok! A kis négyzetek oldalai 20 kp legyenek! A végén 2 másodpercet késleltesd a kilépést!	
9. (05_09.py) Készíts programot egymásba ágyazott ciklusokkal, melyben véletlenszerű helyre (400*400-as területre), véletlenszerű hosszúságú „csillagággal” (1-50 kp). 5 darab csillagot kirajzol 20 ággal 18 fokkal elforgatva! A program lefutása után várjon, amíg a felhasználó bezárja az ablakot!	
10. (05_10.py) Készíts egy programot, melyben egy 10*10 es „mátrix”-ban kiíratsz 1-1000 közötti véletlen számokat, majd a végén a minta szerint kiírja a program, hogy melyik a legnagyobb szám az összes közül! A számokat tabulátorral rendezd a minta szerint!	

11. FELTÉTELES CIKLUSOK

Programozásban gyakran fordulnak elő olyan esetek, **amikor nem tervezhető előre a ciklusok lépésszáma**. Ezekben az esetekben használjuk a **feltételes ciklus**okat úgy, hogy a felhasználótól **bekérünk adatot, majd annak érvényességét vizsgáljuk és csak helyes adatok esetén megy tovább a program, különben újból bekérjük az adatot**. Tehát szükségünk van olyan ciklusra, amiben az ismétlések száma feltételhez kötött. Két fajta ciklust kell megemlítenünk: **elöltesztelő** és hátultesztelő **ciklus**. Mivel a Python csak elöltesztelő ciklust ismeri, ezért erről fogunk tanulni.

Elöltesztelő ciklus

Nézzünk meg egy példát!
A feladat két egész szám **legkisebb közös többszörös**ének megkeresése.

- Például az 5 és a 18 legkisebb közös többszörösét keressük meg!
- Ha elindulunk az 5 többszöröseivel:5, 10,15, 20, 25, 30, … 80, 85, 90,
- A 18 többszöröseivel: 18, 36, 54, 72, 90, (táblázat)
- Majd a program a legkisebb közös többszörösnél megáll. (t1=t2)
- A többszörösöket ügy képezzük, hogy hozzáadjuk az eredeti számot. (+5, +18)
- Nem fogunk egyik, vagy másik szám többszörösével előre rohanni, hanem mindig csak a kisebb többszöröst emeljük, hiszen annak van esélye utolérni a másikat.
- Tehát bekérünk két számot! (sz1, sz2).
- Majd egy t1 és t2 változóban tároljuk a számok többszörösét!
- Ezeket a többszörösöket hasonlítjuk össze egymással: t1>t2.
- Az elöltesztelő ciklusnál a **while** utasítást használjuk.
- Tehát a ciklus elejét a while kulcsszóval jelöljük, mögötte a bennmaradás feltételélt kell megadni, majd a vezérléshez tartozó szokásos kettőspontot adjuk meg.
- A ciklusmagot tagolni kell, beljebb kell kezdeni.

	t1	t2	
+5	5	18	
+5	10	18	
+5	15	18	
	20	18	+18
+5	25	36	
+5	30	36	
+5	35	36	
	40	36	+18
+5	45	54	
+5	50	54	
+5	55	54	
	60	54	+18
+5	65	72	
+5	70	72	
+5	75	72	
	80	72	+18
+5	85	90	
	90	90	

(11a.py)

Mondatszerű leírása a programnak a következő:

be: sz1, sz2,
t1:=sz1
t2:=sz2
ciklus amíg t1=t2
ha t1>t2 akkor t2:=t2+sz2
különben t1:=t1+sz1
elágazás vége
ciklus vége
ki: t1

Az elkészített **python program** a következő:

```
11a.py
1  sz1=int(input("Add meg az egyik számot: "))
2  sz2=int(input("Add meg a másik számot: "))
3  t1=sz1
4  t2=sz2
5  while t1 != t2:
6      if t1>t2:
7          t2=t2+sz2
8      else:
9          t1=t1+sz1
10 print("A legkisebb közös többszötös: ",t1)
```

```
Run:  11a
E:\00_MM\12_Python_programozas\pro
Add meg az egyik számot: 5
Add meg a másik számot: 18
A legkisebb közös többszötös:  90
```

Hátultesztelő ciklus

Írjunk olyan programot, melyben egy kockadobás eredményét kérjük be! Feltételezhetjük, hogy a felhasználó hibás adatot ad meg. A bevitt adat ellenőrzésével ezt szeretnénk elkerülni, ezért a hibás adat bekérése után újból bekérjük az adatot!
Nem tudjuk előre, hogy a felhasználó hányszor fog egymás után hibás adatot bevinni, tehát feltételes ciklus fogunk itt is használni. Az adatbekérésnek mindenképpen le kell futnia egyszer, tehát nincs értelme a ciklus elejére helyezni a feltételt, így **hátultesztelő ciklus**t fogunk alkalmazni.

- Először készítünk egy adatbekérő részt.
- Tehát a kockadobásnál 1-6-ig fogadunk el számokat.
- 1-nél kisebb és 6-nál nagyobb számok esetén újra kérünk egy számot.
- Ezt hátultesztelő ciklusba ágyazzuk.
- Megadjuk a ciklusban maradás feltételét.
- Addig kell ismételni az adatbakérést, amíg a szám nem 1, vagy 1-nél nagyobb és 6, vagy annál kisebb.
- A Python nem használ hátultesztelő ciklust, így a kódot úgy kell kialakítani, hogy elöltesztelőssé alakítjuk. Ehhez a ciklusmagban található utasításokat a ciklus elé is be kell írni a feltételt a ciklus elejére kell helyezni.

(11b.py)

Mondatszerű leírás:

ciklus
ki: „Add meg a kocka dobás eredményét: „
be: x
amíg (x<1) vagy (x>6)
ciklus vége

```
11b.py
1  a=int(input("Add meg a kockadobás eredményét: "))
2  while (a<1) or (a>6):
3      a = int(input("Add meg a kockadobás eredményét: "))
4  print("Az ellenőrzött megadott érték: ",a)
```

```
Run:  11b
C:\Users\kolmank\AppData\Local\Progra
Add meg a kockadobás eredményét: 20
Add meg a kockadobás eredményét: -3
Add meg a kockadobás eredményét: 5
Az ellenőrzött megadott érték:  5
```

Teljes lefedés elve:
Egy program megvalósítása során nagyon sokszor teszteljük a működését, különböző értékekkel.
A változók többfélék lehetnek. A programunkat úgy kell kialakítanunk, hogy az összes lehetőségre gondoljunk, így teljesen lefedjük a programunkat. Erre kiválóan megfelel a mos tanult módszer.

Végtelen ciklusok

Vannak olyan esetek, amelyekben véletlenül, vagy éppen direkt adunk meg olyan feltételt, amelyből semmi képpen nem lehet kilépni. Erre nézzünk egy példát:

(11c.py)

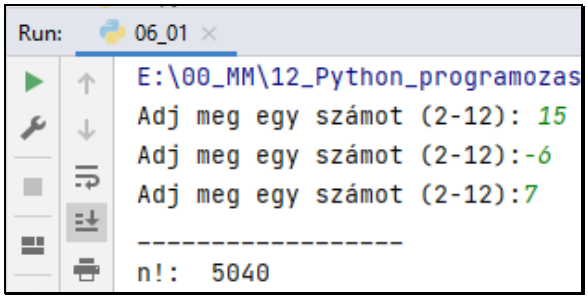
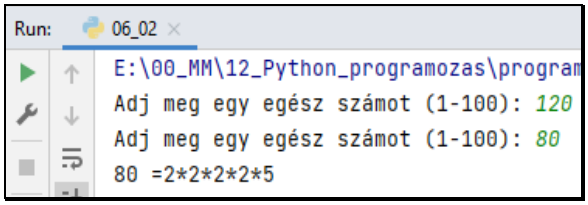
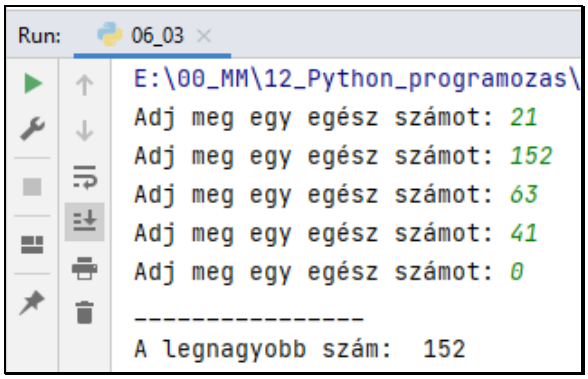
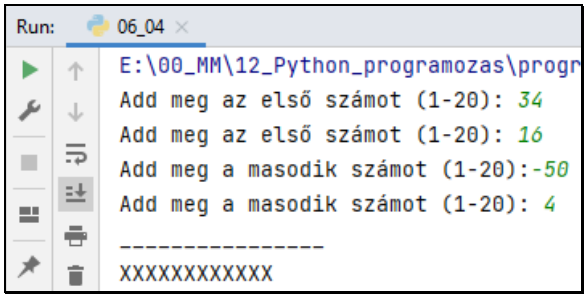
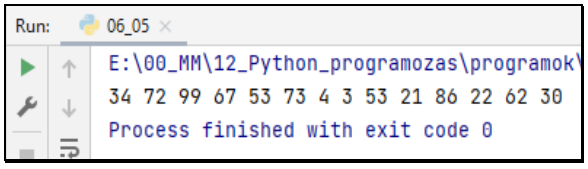
Ebben az egyszerű programban egyértelmű a hiba.
Ilyenkor a futtatáskor nem lép ki soha, csak folyamatosan fut.
Ekkor hiába próbálkozunk kilépni, nem tudunk.
Szoktuk „lefagyott” állapotnak is hívni. Meg kell szakitani a futást.
Egy összetett programban meg kell keresni a hibát és javítani.
A futó Python program megszakítása: Ctrl+F2 bill. kombinációval lehetséges.

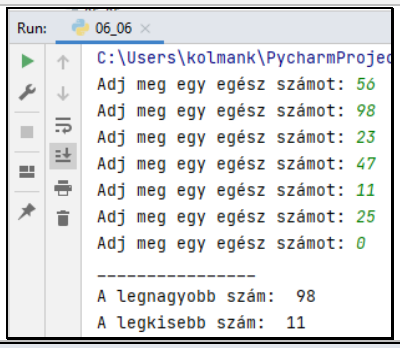
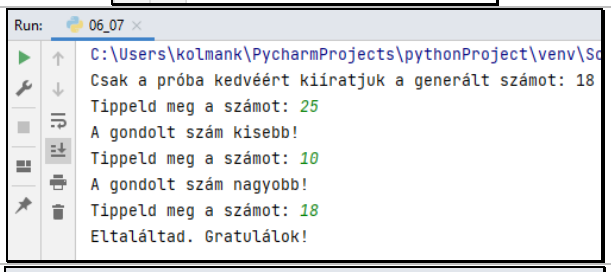
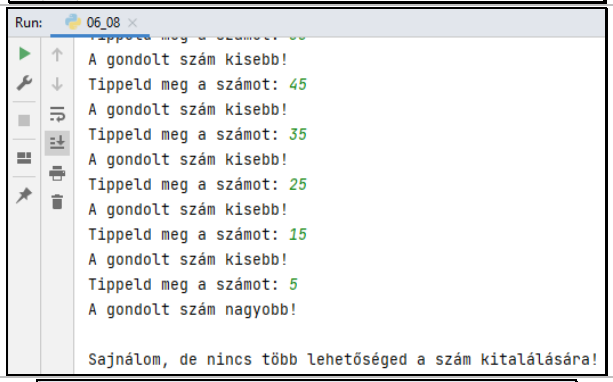
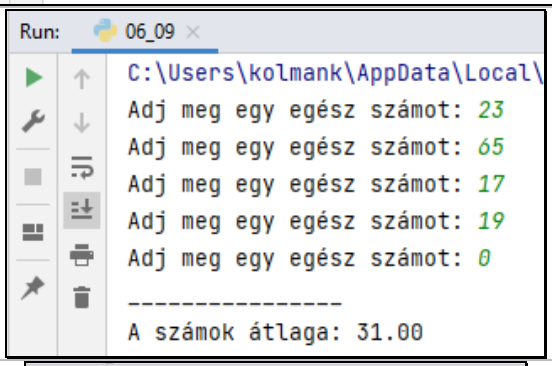
```
11c.py
1  while (1<2):
2      print("Végtelen ciklus!")
```

```
Run:  11c
Végtelen ciklus!
Végtelen ciklus!
Végtelen ciklus!
```

GYAKORLÓ FELADATOK (11. témakör)

Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!
A feladatokat a „while” utasítással old meg!

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (06_01.py) Készíts programot „n” faktoriális kiszámítására (n!). A programot úgy készítsük, hogy el bekér egy 2 és 12 közti számot, majd utána n-től 1-ig szorozzuk össze a számokat egy „szorzat” nevű változóba! A program vizsgálja, hogy jó tartományból legyen a szám!	
2. (06_02.py) Készíts egy programot, amely bekér egy 1 és 100 közötti számot, majd a mintán látható módon kiírja kettes számok szorzataként. Mindaddig osszuk a változót kettővel, amíg már nem osztható és végül kiírjuk azt az utolsó, páratlan számot! A program vizsgálja, hogy jó tartományból legyen a szám!	
3. (06_03.py) Készíts programot, amely bekér egész számokat, mindaddig, míg nem adjuk meg egy 0-t. A program végén határozza meg és írja ki a számok közül a legnagyobbat! Most nem kell vizsgálni a jó tartományt!	
4. (06_04.py) Készíts programot, amely bekér két 0 és 20 közötti egész számot! Majd annyi nagy „X” karaktert írki egy vonal alá, amennyi a két szám különbsége! A számok bekérésénél használj „while” ciklust – a teljes lefedés elve szerint -, a különbségnél pedig abszolút függvényt használj!	
5. (06_05.py) Készíts programot, amely véletlen egész számokat generál 1 és 100 között mindaddig, amig véletlenül 10-el osztható egész számot nem generál! A számoka t egymás mellé írasd ki!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
6. (06_06.py) Bővítsd ki az előző oldalon lévő 06_03.py programot, és mentsd el másként 06_06.py néven úgy, hogy a legnagyobb szám kikeresése mellé a legkisebb számot is kiírja! Figyelj arra, hogy a nulla (amivel befejezteted az adatbevitelt) az ne számítson be az értékek közé!	
7. (06_07.py) Készítsél programot, melyben a számítógép „gondol” egy 1 és 50 közötti számra. Magyarul generál egy véletlen számot 1 és 50 között! Majd tippeket kér be, és kiírja, hogy nagyobb, vagy kisebb a szám, mindaddig, amíg el nem találjuk!	
8. (06_08.py) Bővítsd ki az előző programot, és mentsd el másként! Úgy változtasd meg a programot, hogy csak 7-szer tippelhessen a felhasználó! Ha nem sikerült kitalálni annyi próbálkozás alatt, akkor írja ki, hogy nem találta el, és írja ki a megfejtést is!	
9. (06_09.py) Készíts programot, melyben pozitív egyész számokat kér be 1 és 100 között 0 végjelig! Majd kiírja a számok átlagát! A nullát ne számolja bele az átlagba!	
10. (06_10.py) Készíts programot, amely ki fogja kérdezni a matematikát (két szám összeadását, az <1,10> intervallumból). A két számot a számítógép véletlenszerűen válassza ki. A program akkor fejeződjön be, ha a felhasználó 10 példát kiszámolt helyesen. Rossz válasz esetén kérdezze újra ugyanazt a példát. A program végén írjuk ki az eredményességet százalékokban	

12. KARAKTEREK A PYTHON PROGRAMOZÁSBAN

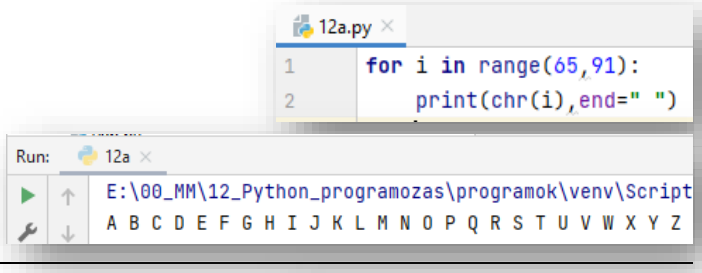
A programozásban sokszor előfordul, hogy karakterekkel kell dolgoznunk. A karakterek ASCII kódjuk, bájt nagyságú számok formájában tárolódnak. A lenti táblázatban egyértelműen látjuk, hogy melyik karakterhez melyik szám tartozik.
Megkaphatjuk **egy karakter ASCII kódját** az **ord** utasítással. Pl.: A➔65, a➔97,
Egy beírt **számból a megfelelő ASCII karaktert** a **chr** utasítással állítjuk elő. pl.: 90➔Z, 122➔z

ASCII kódtábla (32-255):

32=	33=!	34="	35=#	36=$	37=%	38=&	39='	40=(	41=)
42=*	43=+	44=,	45=-	46=.	47=/	48=0	49=1	50=2	51=3
52=4	53=5	54=6	55=7	56=8	57=9	58=:	59=;	60=<	61==
62=>	63=?	64=@	65=A	66=B	67=C	68=D	69=E	70=F	71=G
72=H	73=I	74=J	75=K	76=L	77=M	78=N	79=O	80=P	81=Q
82=R	83=S	84=T	85=U	86=V	87=W	88=X	89=Y	90=Z	91=[
92=\	93=]	94=^	95=_	96=`	97=a	98=b	99=c	100=d	101=e
102=f	103=g	104=h	105=i	106=j	107=k	108=l	109=m	110=n	111=o
112=p	113=q	114=r	115=s	116=t	117=u	118=v	119=w	120=x	121=y
122=z	123={	124=|	125=}	126=~	127=•	128=€	129=▯	130=‚	131=*ƒ*
132=„	133=…	134=†	135=‡	136=ˆ	137=‰	138=Š	139=‹	140=Œ	141=▯
142=Ž	143=▯	144=▯	145=‘	146=’	147=“	148=”	149=•	150=–	151=—
152=˜	153=™	154=š	155=›	156=œ	157=▯	158=ž	159=Ÿ	160=	161=¡
162=¢	163=£	164=¤	165=¥	166=¦	167=§	168=¨	169=©	170=ª	171=«
172=¬	173=	174=®	175=¯	176=°	177=±	178=²	179=³	180=´	181=µ
182=¶	183=·	184=¸	185=¹	186=º	187=»	188=¼	189=½	190=¾	191=¿
192=À	193=Á	194=Â	195=Ã	196=Ä	197=Å	198=Æ	199=Ç	200=È	201=É
202=Ê	203=Ë	204=Ì	205=Í	206=Î	207=Ï	208=Ð	209=Ñ	210=Ò	211=Ó
212=Ô	213=Õ	214=Ö	215=×	216=Ø	217=Ù	218=Ú	219=Û	220=Ü	221=Ý
222=Þ	223=ß	224=à	225=á	226=â	227=ã	228=ä	229=å	230=æ	231=ç
232=è	233=é	234=ê	235=ë	236=ì	237=í	238=î	239=ï	240=ð	241=ñ
242=ò	243=ó	244=ô	245=õ	246=ö	247=÷	248=ø	249=ù	250=ú	251=û
252=ü	253=ý	254=þ	255=ÿ						

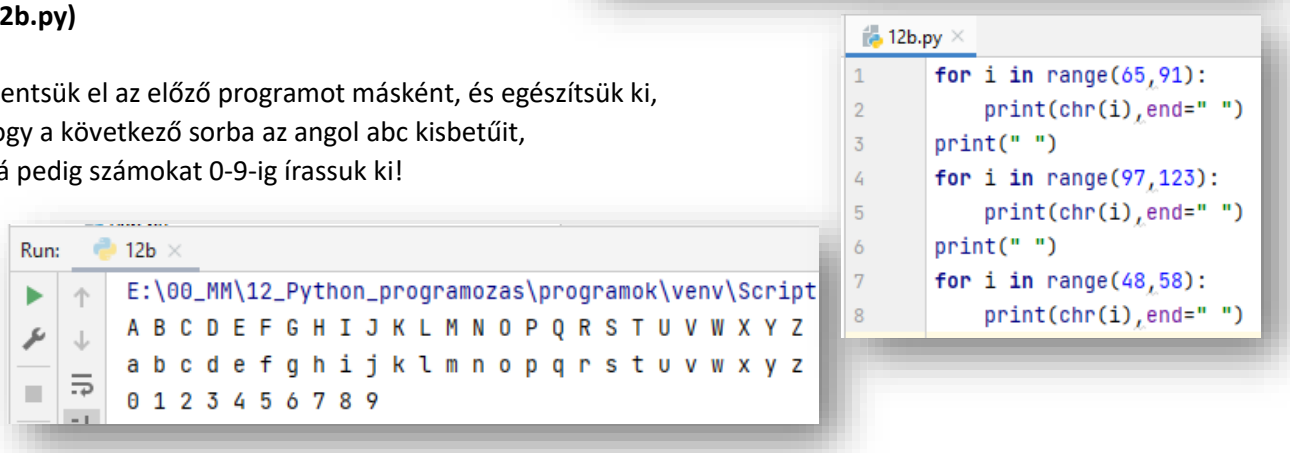
(12a.py)

Írassuk az angol ABC nagybetűit egymás mellé!



(12b.py)

Mentsük el az előző programot másként, és egészítsük ki,
hogy a következő sorba az angol abc kisbetűit,
alá pedig számokat 0-9-ig írassuk ki!



(12c.py)

Készítsünk programot, melyben egy begépelt karakterről eldöntjük, hogy nagybetű-e vagy nem!

```
12c.py
1  betu=input("Adj meg egy karaktert: ")
2  if "A"<=betu<="Z":
3      print("A karakter nagybetű.")
4  else:
5      print("A karakter nem nagybetű.")
```

```
Run: 12c
E:\00_MM\12_Python_program
Adj meg egy karaktert: F
A karakter nagybetű.
```

(12d.py)

Készítsünk programot, mellyel régi magyarországi rendszámot állítunk elő véletlenszerű karakterekből és számokból.

```
Run: 12d
E:\00_MM\1
DCI-480
```

```
12d.py
1  from random import *
2  x1=randrange(65,91)
3  x2=randrange(65,91)
4  x3=randrange(65,91)
5  sz1=randrange(0,10)
6  sz2=randrange(0,10)
7  sz3=randrange(0,10)
8  n1=chr(x1)
9  n2=chr(x2)
10 n3=chr(x3)
11 print(n1,n2,n3,"-",sz1,sz2,sz3, sep="")
```

(12e.py)

Általában decimális (tízes) számrendszert használunk a mindennapokban. Informatikában a decimális (kettes) számrendszer mellett a hexadecimális (tizenhatos) számrendszert is használjuk. Ebben 0-9-ig és A-F-ig használunk számokat és karaktereket.
Készítsünk programot, melyben bekérünk egy hexadecimális számot, majd kiírjuk a decimális értékét!

```
12e.py
1  hexa=input("Kérek egy 0-F karaktert: ")
2  if hexa <= '9':
3      dec=ord(hexa)-48
4  else:
5      dec=ord(hexa)-55
6  print("Tízes számrendszerben: ",dec)
```

```
Run: 12e
E:\00_MM\12_Python_programoz
Kérek egy 0-F karaktert: B
Tízes számrendszerben:  11
```

(12f.py)

A feladat az, hogy kérjen be egy szót, majd karakterenként tabulátorral elválasztva írassuk ki a betűket!

```
12f.py
1  szo=input("Írjál be egyszót: ")
2  hossz=len(szo)
3  for i in range(hossz):
4      print(szo[i],end="\t")
```

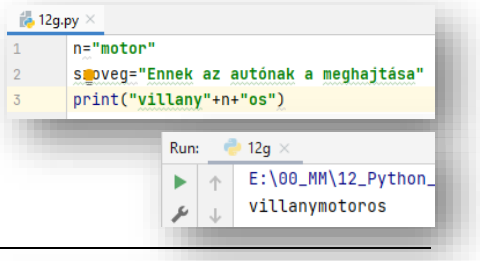
```
Run: 12f
E:\00_MM\12_Python_programozas\progra
Írjál be egyszót: Háromszög
H   á   r   o   m   s   z   ö   g
```

- Kérjünk be egy „szo” változóba egy szót (karakterláncot)!
- Majd a **len() utasítással határozzunk meg a szó karaktereinek számát**! (Milyen hosszú a karakterlánc.)
- For utasítással 0-tól hossz karakterszámig megy a ciklus!
- Majd kiíratjuk a szó „i”-edik karakterit, tabulátorokkal elválasztva, a minta szerint.
- A „szo[i]” arra szolgál, hogy egyesével végig lépkedünk karakterenként.
- Az **end=”\t”** utasítással tabulátorokkal tagoljuk a karaktereket:

Ahogyan már többször is használtuk, a karaktereket idézőjelek, vagy aposztrófok közé kell írni. (pl.: „villanymotor”, ’villanymotor’).

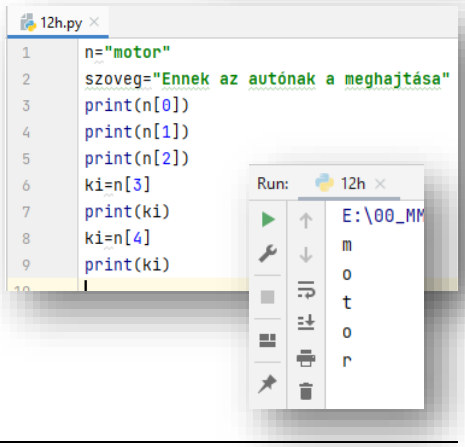
(12g.py)

Nézzünk egy példát, amikor idézőjelben megadott szöveget és változóban tárolt **szöveget fűzünk össze plusz (+) műveleti jelekkel**!



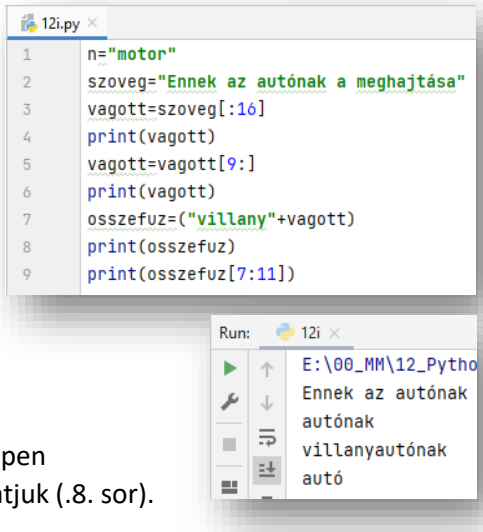
(12h.py)

Ha hivatkozni szeretnének egy karakterlánc valahányadik karakterére akkor az **adott változó neve után négyzetes zárójelben megaduk, hogy hányadik karaktert szeretnénk felhasználni.**
Arra figyelni kell, hogy mindig nulla értéktől számolunk. Tehát az „n” változó első betűje az az **n[0]**, és print() utasítással ki tudjuk íratni közvetlenül.
A példánkban az ”n” harmadik karakterét beletesszük egy „ki” nevű változóba és az íratjuk ki!



(12i.py)

Ebben a példában a változónkban tárolt **karakterláncunk elejéről és végéről „vágunk” le felesleges részeket.**
A 3. sorban a „szoveg” változóban lévő karakterlánc első 16+1 karakterét tesszük bele a „vagott” változóba.
Amit a következő 4. sorban kiíratunk. (Ez a futtatásunk első sora.)
Majd az 5. sorban az éppen aktuálisan a „vagott” változóban lévő stringláncból (amit az előbb kiírattunk) használja fel a 9. karakter utáni részt, és most ez kerül be a „vagott” nevű változóba! **Fontos tehát, hogy a kettőpontokat hova tesszük!**
A 7. sorban egy „osszefuz” változóba összefűzzük a „villany” és az éppen aktuálisan a „vagott” változóban lévő „autónak” szöveget. Majd kiíratjuk (.8. sor).
A 9. sorban látható módon pedig karakterszámtól, karakterszámig írathatunk ki szöveget.

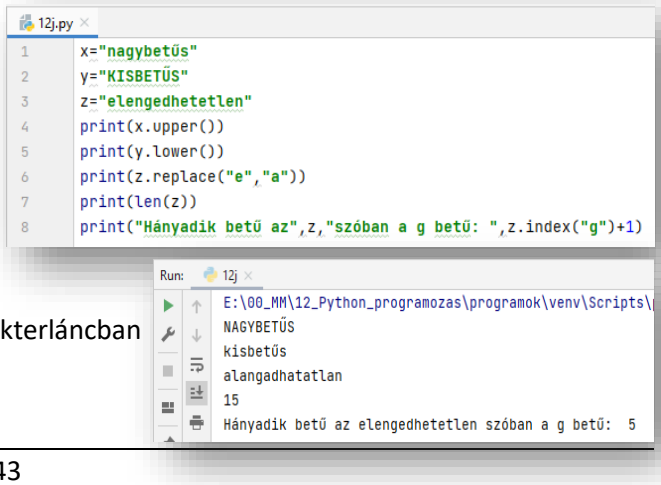


(12j.py)

Végül egyetlen programban nézzünk meg több hasznos utasítást:

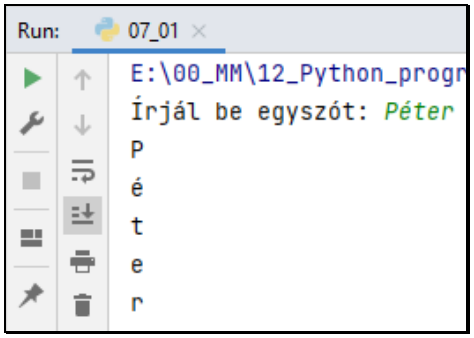
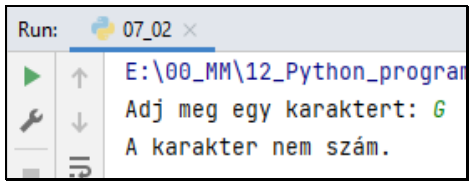
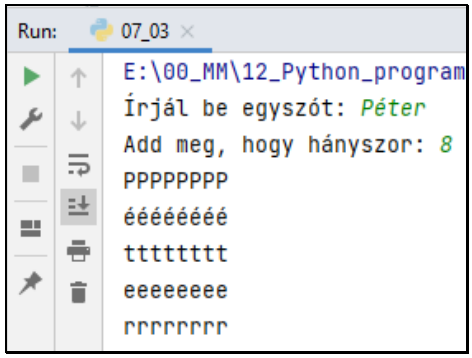
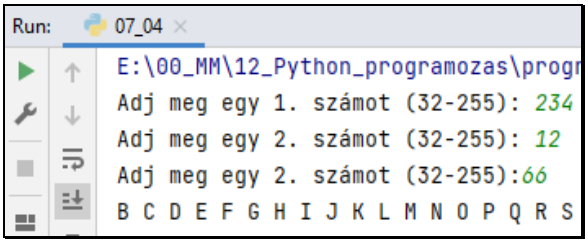
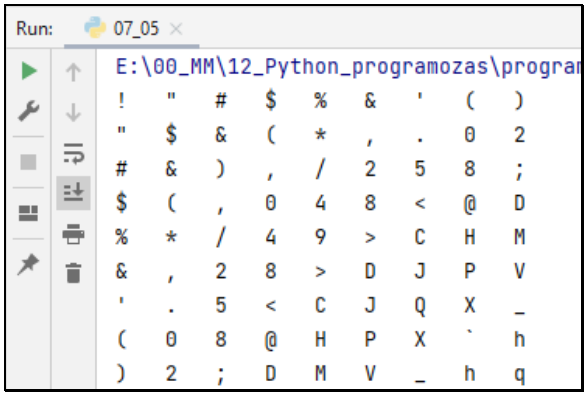
- **upper()** – nagybetűssé teszi a karakterláncot
- **lower()** – kisbetűssé teszi a karakterláncot
- **replace(’x’,’y’)** – cseréli a karkterket
- **len(z)** – visszaadja a karakter hosszát
- **index(’n’)** – megkeresi az első előfordulását a karakterláncban

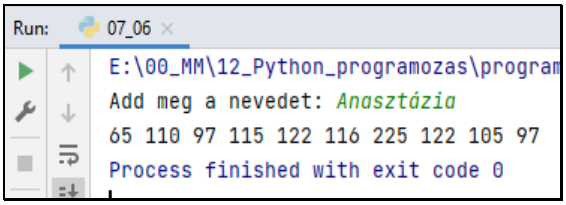
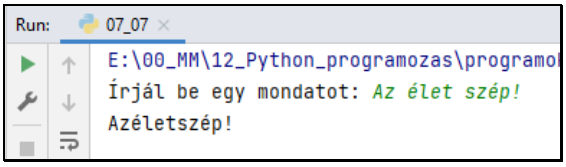
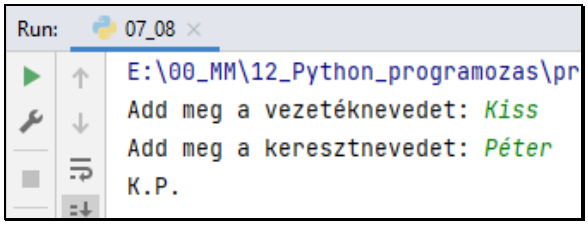
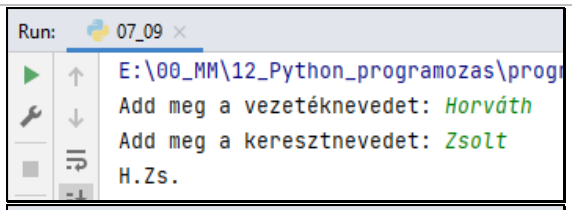
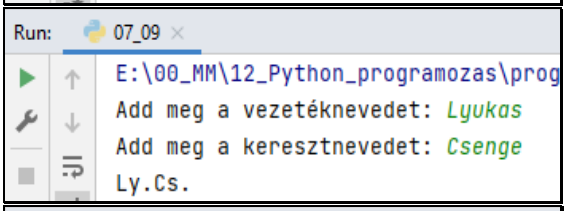
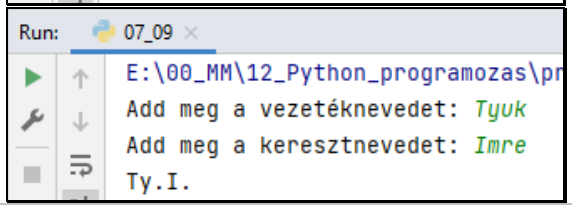
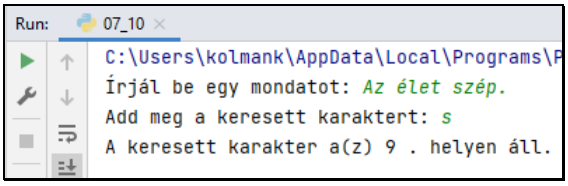
A futtatott programban látszik az eredmény.



GYAKORLÓ FELADATOK (12. témakör)

Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

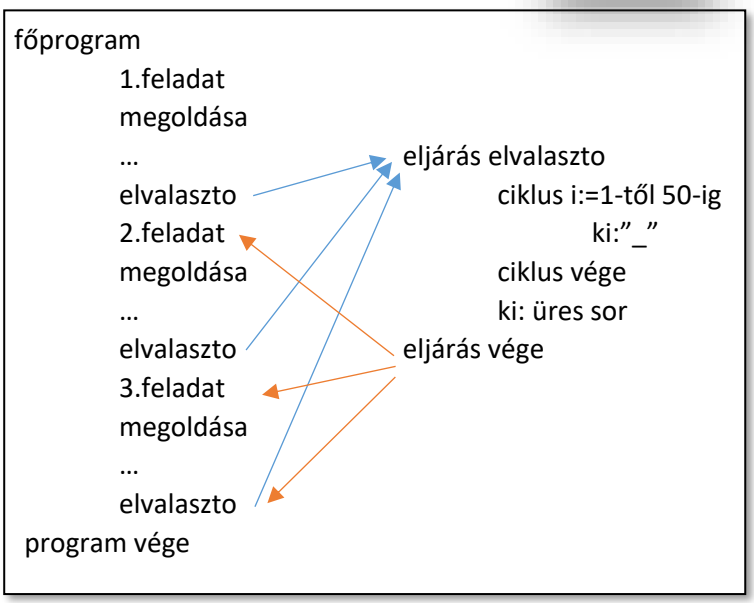
FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (07_01.py) Készítsél egy programot, melyben bekér egy szót az elején, majd kiírja a szó karaktereit egymás alá külön sorokba!	
2. (07_02.py) Készíts egy programot, amely bekér egy karaktert, melyről eldönti, hogy számjegy-e, vagy nem az!	
3. (07_03.py) Készíts programot, mely bekér egy szót, majd bekér egy számot! Majd írasd ki a minta szerint a szó első karakterét annyiszor, amekkora számot megadtál! Majd a következő sorba a második karaktert, és így tovább!	
4. (07_04.py) Készíts egy programot, amely két számot kér be 32 és 255 között! Ha nem megfelelő számot írunk be, akkor kérjük be újból a számot! Ha beírtuk a megfelelő számokat, akkor írjuk ki a két szám közti ASCII karaktereket egymás mellé szóköz távolságra a minta szerint!	
5. (07_05.py) Írassuk ki a tejes ASCII kódtáblát 10 oszlopba és 23 sorban a minta szerint! A karakterek tabulátorral legyenek tagolva!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
6. (07_06.py) Készíts programot, amely bekér egy nevet, majd kiíratja egy sorba a karakterek ASCII kódját a minta szerint!	
7. (07_07.py) Készíts egy programot, amely bekér egy mondatot, és kiszedi belőle a szóközöket, majd kiíratja a képernyőre a minta szerint!	
8. (07_08.py) Készíts programot, mely bekér egy vezetéknevet és egy keresztnevet, majd kiírja a monogramot, pontokkal elválasztva a minta szerint!	
9. (07_09.py) Mentsd el az előző programot másnéven! Majd változtasd meg úgy, hogyha vakinek kétjegyű mássalhangzóval kezdődik vagy a vezetékneve vagy a keresztneve, azt kezelje a program! A példákon látható módon kezelje a neveket!	  
10. (07_10.py) Készíts egy programot, melybe be kell gépelni egy mondatot, majd a következő sorban kér egy karaktert! A program írja ki azt, hogy hányadik helyen van az első előfordulása a beírt karakternek!	

13. ELJÁRÁSOK (PROCEDURE)

Az eljárások (és függvények) tulajdonképpen **alprogramok, vagy más néven részprogramok**. **Az alprogramokat névvel látjuk el, majd a főprogramban a nevével hivatkozunk rá és végrehajtódik.**

- Az eljárások és függvények előnye a kódismétlés elkerülése.
- A programunk rövidebb, áttekinthetőbb lesz.
- Logikusabban építhetjük fel. Többször felhasználhatunk alprogramokat.
- Elég az alprogramon változtatni, ha kód hibás, vagy éppen másképp szeretnénk működtetni.
- Nagyobb program csoportmunkában fejleszthető.
- Használatakor megváltozik az alapvető sorrendi végrehajtás.



Nézzük a jobb oldali mondatszerű leírást az **eljárások meghívásá**ra:

- A főprogram elindulása után számozott feladatokat hajt végre.
- A baloldalon a főprogramot, a jobb oldalon az alprogramot látjuk.
- A képernyőre- való kiírásnál a feladatokat ötven darab „_" karakter vonallal szeretnénk elválasztani.
- Azért, hogy ne kelljen ugyanazt, többször begépelni, bemásolni, akkor eljárást használunk.
- Készítünk egy eljárást, amit meghívhatunk, ahányszor szeretnénk, egyszerűen csak a nevével.
- Aztán mindig visszatér a főprogramba.

```
eljárás alprogram2
…
eljárás vége

eljárás alprogram1
        alprogram2
        …
eljárás vége

főprogram
        alprogram1
```

Az eljárások **lefutásának sorrendje**:

- Arra mindig figyelni kell, hogy amely eljárást felhasználunk, azt mindenképpen előbbre definiálni kell!
- A jobb oldali példában látni, hogy a főprogram lefutásakor az alprogram1-et hívja meg, amelyben az alprogram2 van meghívva. Tehát az alprogram2-nek előbbre kell lennie mint az alprogram1-nek.

Eljárás paraméter nélkül

(13a.py)

- Az eljárás létrehozását a **def kulcsszóval kezdünk**
- **Ezt követi az eljárás neve, majd egy üres zárójelpár** (ha nincsenek átadandó paraméterek) és **végül kettőspont**tal zárul a sor.
- Az eljárás **tartalmi rész**ét egy bekezdéssel beljebb kell kezdeni.
- A **tagolás vége jelzi az eljárás végét**.
- Az eljárást a főprogramból a **nevével és az üres zárójelpárral hívjuk** meg.
- A főprogramban ahányszor szeretnénk, annyiszor hívhatjuk meg az eljárást.

13a.py

```
1  def elvalaszto():
2      for i in range(60):
3          print("_",sep=" ",end=" ")
4      print("\n")
5  print("1. feladat: ")
6  elvalaszto()
7  print("2. feladat: ")
8  elvalaszto()
9  print("3. feladat: ")
10 elvalaszto()
```

Eljárás paraméterátadással

(13b.py)

A jobb oldali példán látjuk a paraméterek megadását:

- Az eljárások működését paraméterek segítségével befolyásolhatjuk.
- Az előző példa folytatásaként / kibővítéseként megadhatjuk, hogy milyen széles legyen a vonal, és azt is, hogy milyen karakterből legyen megrajzolva.
- A paramétereket a zárójelek között adjuk meg vesszővel elválasztva egymástól.
- Ezeket a paramétereket felhasználjuk az eljárás megfelelő részében.
- Értelmezzük a mintaprogramot!

```
13b.py
1  def elvalaszto(db,k):
2      for i in range(db):
3          print(k,sep=" ",end=" ")
4      print("\n")
5  print("1. feladat: ")
6  elvalaszto(30,"*")
7  print("2. feladat: ")
8  elvalaszto(20,"_")
9  print("3. feladat: ")
10 elvalaszto(10,"/")
```

```
Run:  13b
E:\00_MM\12_Python_programozas
1. feladat:
* * * * * * * * * * * * * * *

2. feladat:
- - - - - - - - - - - - - - -

3. feladat:
/ / / / / / / / / /
```

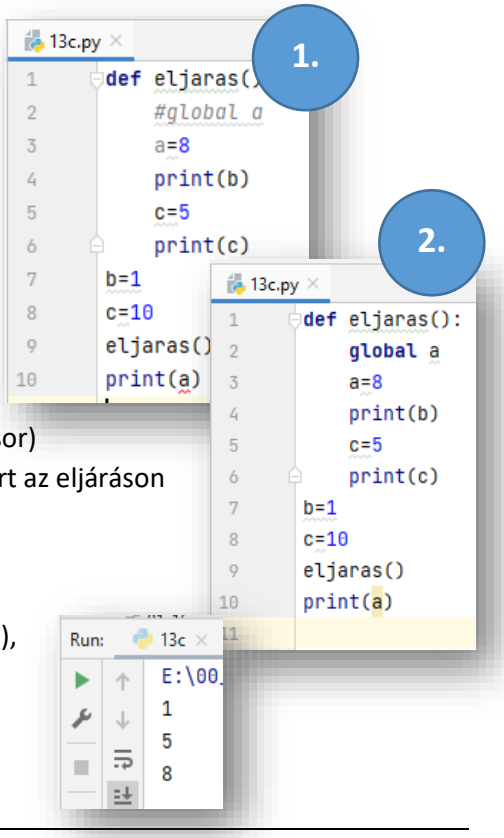
Változók hatásköre

Fontos tudni, hogy **az alprogramokban létrehozott változók, csak az eljáráson belül használhatók**. (Ha erről másképpen nem rendelkezünk.) Hiába hivatkozunk rá a főprogramban, vagy egy másik eljárásban, csak hibaüzenetet kapunk.
A **modulokban létrehozott változókat lokálisnak nevezzük**. Csak a modulban van érvényességi körük.
A **főprogramban létrehozott változókat globálisnak nevezzük**.
Érvényességi körük a teljes program, ha nincs azonos nevű lokális változó!
A Pythonban van lehetőség, hogy modulban is létrehozhatunk globális változót, ha global kulcsszót használjuk.

(13c.py)

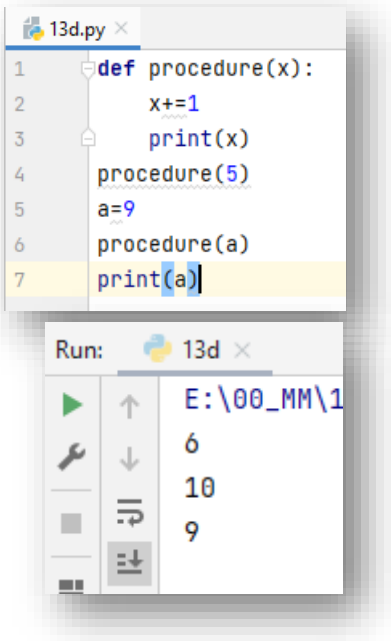
Nézzük a **változók használat**át alaposabban:

- Az „a” változót az 1. képen (3. sor) az eljáráson belül hozzuk létre, csak ott használjuk. Ez egy lokákis változó. Ezért a program lefutásakor, amikor ki akarjuk íratni (10. sor) hibát fog jelezni.
- A főprogramban létrehozott változók alapvetően a teljes programban használhatók.
- A programban létrehozunk egy „b” változót 1 értékkel (7. sor) és egy „c” változót 10 értékkel (8.sor).
- Egy „c” nevű változót az eljárásban is létrehozunk, 5 értékkel. (5. sor)
- Ebben az esetben a lokális változóértékét fogja megjeleníteni, mert az eljáráson belüli megjelenítés következik utána (6.sor).
- Ha az 5. sort kivesszük az eljárásból például # karakter eléírásával, akkor a c értékként a 10 fog megjelenni.
- Ha kivesszük a 2. sorban a # karaktert a global utasítás elől (2. kép), akkor az „a” kiíratásánál meg fog jelenni a 8 érték



Érték szerinti paraméterátadás

(13d.py)



Az előző oldalon lévő felső feladatban (13b.py) megismert módszerben, a főprogramból csak a változó értéke kerül át az alprogramba.

Ebben az esetben pedig nézzük a program lefutását részletesen:

- A 4. sorban indul a program lefutása a „procedure” alprogram meghívásával. Ahol 5-ös értéket ad át.
- A program végrehajtása az első sorban folytatódik, ahol az 5-ös az x változóba kerül, majd a második sorban eggyel növekszik, tehát 6 lesz. Ezt a 6-os értéket fogja a következő, harmadik sor miatt kiírni a képernyőre.
- Aztán az alprogramból kilépve, visszatér a főprogram ötödik sorába, ahol egy „a” nevű változó kap egy 9-es értéket.
- A hatodik sorba lépve meghívjuk újból az alprogramot, most az „a” változót beküldve.
- Így újra az első sorba ugorva, az „a” értéket használva kerül az „x” helyére, így eggyel növelve az érték már 10 és ez kerül kiíratásra a következő sorban.
- Visszaugrunk a főprogramba, ahol az eredetileg megadott „a” értéket kiíratjuk a képernyőre.

Minden esetben a logikáját kell megérteni a feladatnak, hogy mi, miért, és hogyan történik. Hogyan adunk át értéket az alprogramnak, az mit kezd vele, mi történik, amikor visszatér a főprogramba.

(13e.py)

A feladatban grafikus módban meg kell rajzolni egy magyar zászlót! Mivel a zászló három darab vízszintes téglalapból áll, ezért alprogramként definiáljunk egy „teglalap” nevű eljárást, amely rajzol egy 100*400 képpontos téglalapot! Majd a kitöltőszínek változtatásával és a kiindulópont áthelyezésével rajzoljuk meg a zászlót!

Nézzük a lépéseket:

- Meghívjuk a rajzoláshoz szükséges „turtle” eszközt.
- Definiálunk egy olyan eljárást, amely megrajzol egy téglalapot. Felhasználjuk az eljárás elején a téglalapok kitöltéséhez szükséges utasítást!
- A főprogramban először letöröljük a képernyőt a „reset()” utasítással.
- Majd megadjuk a rajzolószínt a „color()” utasítással.
- Aztán az első kitöltőszínt adjuk meg a „fillcolor()” utasítással.
- Meghívjuk az eljárásunkat.
- Felemeljük a tollat. up()
- Elpozícionálunk a 100 kp-al arrébb, ahova rajzolni kell a második téglalapot, és irányba állunk. Aztán letesszük a tollat. down()
- Majd a következőkben megrajzoljuk a másik két téglalapot a megváltoztatott színekkel és át pozícionált kiinduló pontokkal, a minta szerint!
- Teszteljük a programot közben és javítsuk az esetleges hibákat!!

(13f.py)

Rajzoljunk egy más mellé, egy magyar és egy német zászlót!

Alakítsuk át az előző programot:

- Úgy, hogy egy újabb eljárásban – melynek a neve legyen tricolor - a három sávos zászlót is megrajzoljuk.
- A teglalap() eljárásnak előbbre kell lennie, mit a tricolor() eljárásnak, mert a tricolorban felhasználjuk a teglalap eljárást.
- A tricolor nevű eljárásnak legyen három átadandó értéke a f(első), k(özépső), és a(lsó)!
- A tricolor() eljárásban megrajzoljuk, pozícionáljuk és színezzük a három téglalapot.
- A főprogramban törlés után megadjuk a rajzolószínt, ami ebben az eseteben „arany”.
- Majd meghívjuk a tricolor(f,k,a) eljárást, ahol a változók helyére megadjuk a színek nevét angolul, idézőjelek közé írva.
- Majd felemeljük a „tollat” és úgy pozícionálunk jobbra, hogy a két zászló között 20 képpont legyen.

```
from turtle import *
def teglalap () :
    begin_fill()
    forward(300)
    left(90)
    forward(100)
    left(90)
    forward(300)
    left(90)
    forward(100)
    end_fill()

def tricolor(f, k, a):
    fillcolor(f)
    teglalap()
    up()
    forward(100)
    left(90)
    down()
    fillcolor(k)
    teglalap()
    up()
    forward(100)
    left(90)
    down()
    fillcolor(a)
    teglalap()
    left(90)

reset()
color("gold")
tricolor("red","white","green")
up()
goto(-320,0)
down()
tricolor("black","red","yellow")
```


```
from turtle import *
def teglalap () :
    begin_fill()
    forward(100)
    left(90)
    forward(300)
    left(90)
    forward(100)
    left(90)
    forward(300)
    end_fill()

def ftricolor(f, k, a):
    fillcolor(f)
    teglalap()
    up()
    left(90)
    forward(100)
    down()
    fillcolor(k)
    teglalap()
    up()
    left(90)
    forward(100)
    down()
    fillcolor(a)
    teglalap()
    left(90)
reset()
color("gold")
goto(0,-150)
ftricolor("green","white","orange")
up()
goto(-320,-150)
down()
ftricolor("green","white","red")
```

(13g.py)

A következő feladatban függőlegesen három csíkos zászlót rajzoltatunk ki egymás mellé.

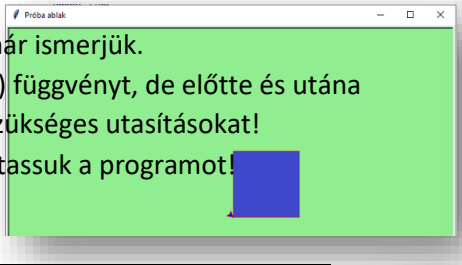
- A kész zászlók mérete legyen 300*300 képpontos!
- Ennél a feladatnál a logika ugyan az, mint az előzőekben, csak a téglalap méretét és irányát kell újra gondolni, meg a pozicionálásukat.
- Mindig figyeljünk arra, hogy mikor kell felemelni a „tollat”, mikor kell letenni!
- Hova, milyen irányba rajzoljuk meg a téglalapokat!

(13h.py)

Ebben a feladatban a grafikus megjelenítésre nézünk egy példát, és megnézünk pár új utasítást!

A feladat egy egyszerű téglalap kirajzoltatása!

- Hozzunk létre egy negyzet() nevű eljárást, mely megrajzol egy négyzetet!
- A **bgcolor(„lightgreen”)** parancs **háttérszínt ad az ablaknak**!
- A **title(„Próba ablak”)** utasítással, az **ablak címsorában megjelenő szöveget adhatunk meg**!
- A color() és fillcolor() utasítást pedig már ismerjük.
- A főprogramban meghívjuk a négyzet() függvényt, de előtte és utána megadjuk a zárt alakzat kitöltéséhez szükséges utasításokat!
- A kilépés előtt 3 másodpercet várakoztassuk a programot!



14e.py

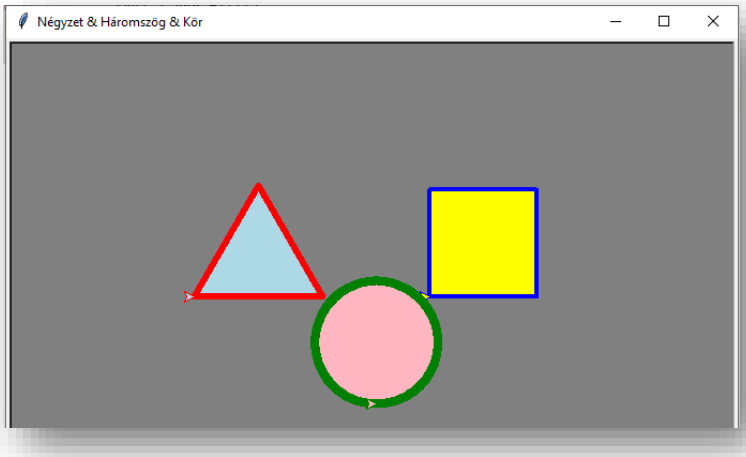
```
import time
from turtle import *
def negyzet(h):
    for i in range(4):
        forward(h)
        left(90)

bgcolor("lightgreen")
title("Próba ablak")
color("red")
fillcolor("blue")

begin_fill()
negyzet(100)
end_fill()

time.sleep(3.0)
```

(13i.py)

Példányok (több teknőc egy programban)

Mint ahogy sok különböző egész változónk is lehet egy programban, úgy sok „teknőcünk” is lehet egyszerre. Mindegyik egy ún. **példány**. Minden egyes példánynak saját tulajdonságai vannak és saját metódusai – így a teknos_1 rajzolhat egy vékony kék tollal egy bizonyos pozícióban, míg teknos_2 haladhat a saját útján egy vastag piros tollal, stb… .

- Az ismert módon az első sor megmondja a Pythonnak, hogy töltse be a „turtle” nevű modult. Amely lehetővé teszi a grafikus felület használatát.
- Majd a 2-4. sorig beállítjuk az ablak tulajdonságát!
- Majd létrehozzuk 6-9 sorokban a teki_1, 11-14-ig a teki_2, és a 16-19-ig a teki_3-mat a tulajdonságaival! (rajzolószín, ceruza vastagsá
- Aztán definiálunk először egy négyzet(a) függvényt, amelyet pozícionálunk, felemelt ceruzával, majd megrajzolunk egy négyzetet, kitöltőszínnel!
- Így teszünk utána a háromszög(a) nevű függvénnyel.
- És a kör(r) függvénnyel is.
- A főprogramban csak meghívjuk a függvényeket a megadott értékekkel a minta szerint!
- Végül az utasítás, ami vár az ablak bezárására. (**win.mainloop()**)

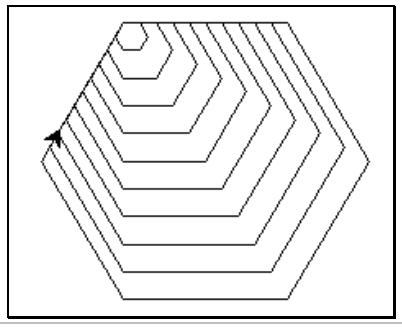
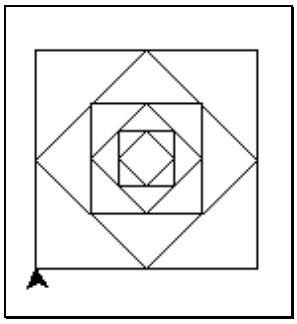
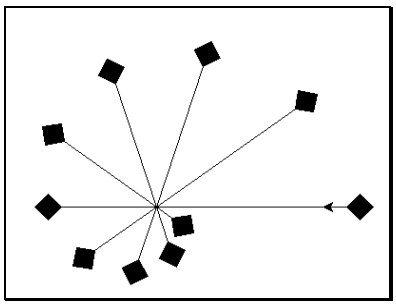
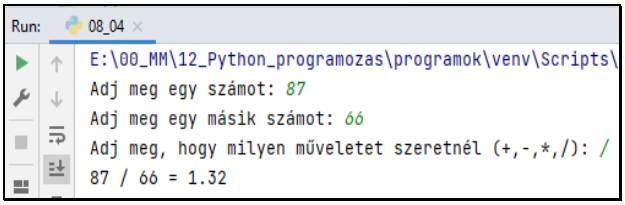
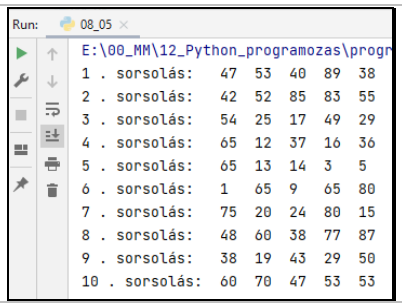


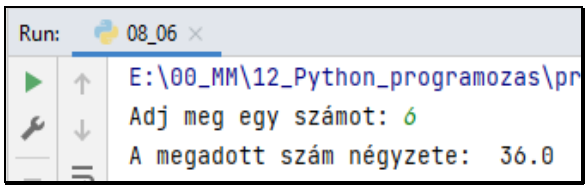
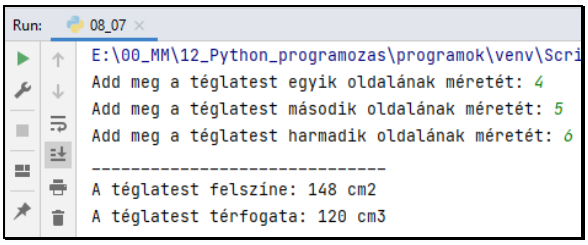
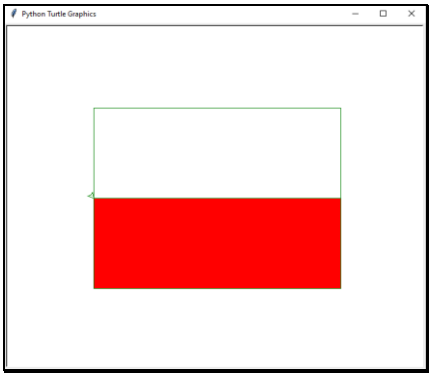
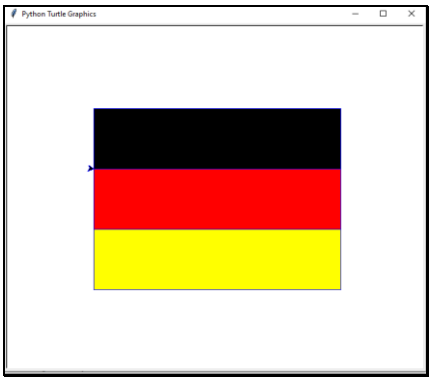
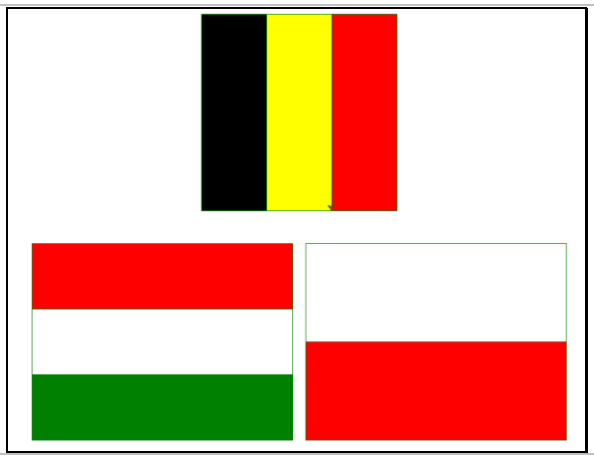
14_f.py

```
import turtle
win = turtle.Screen()
win.bgcolor("gray")
win.title("Négyzet & Háromszög & Kör")

teki_1 = turtle.Turtle()
teki_1.color("blue")
teki_1.pensize(4)
teki_1.fillcolor("yellow")

teki_2 = turtle.Turtle()
teki_2.color("red")
teki_2.pensize(6)
teki_2.fillcolor("lightblue")

teki_3 = turtle.Turtle()
teki_3.color("green")
teki_3.pensize(8)
teki_3.fillcolor("lightpink")

def negyzet(a):
    teki_1.up()
    teki_1.goto(50, 50)
    teki_1.down()
    teki_1.begin_fill()
    for i in range(4):
        teki_1.forward(a)
        teki_1.left(90)
    teki_1.end_fill()

def haromszog(a):
    teki_2.up()
    teki_2.goto(-170, 50)
    teki_2.down()
    teki_2.begin_fill()
    for i in range(3):
        teki_2.forward(a)
        teki_2.left(120)
    teki_2.end_fill()

def kor(r):
    teki_3.up()
    teki_3.goto(0, -50)
    teki_3.down()
    teki_3.begin_fill()
    for i in range(360):
        teki_3.forward(r)
        teki_3.left(r)
    teki_3.end_fill()

negyzet(100)
haromszog(120)
kor(1)

win.mainloop()
```

GYAKORLÓ FELADATOK (13. témakörhöz)

Készítsed el a következő feladatokat utasítások alapján! A végeredmény kinézete mindig a képernyőképpel megegyező legyen! A programok neve, a feladat elején zárójelben lévő szám legyen, minden esetben!

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (08_01.py) Készíts programot, melyben kirajzolod a jobb oldalon lévő alakzatot! A programban hozzál létre egy „elem()” nevű eljárást, amelyet a főprogramban meghívsz! A hatszög oldalát 10 kp-ról indítsd, melyet mindig 10 kp-al növelsz! 10 darab hatszöget rajzolj ki! A kirajzolás után 2 másodpercet várjon, mielőtt a futó programot bezárja!	
2. (08_02.py) Készíts programot a következők alapján, a jobb oldali minta szerint! A program elején definiálj egy „elem()” nevű eljárást, mellyel megrajzolsz egy négyzetet! Majd hatszor meghívod ezt az eljárást úgy hogy változtatod a oldalának méretét! Az első négyzet oldalának a mérete legyen 20 kp! De arra figyelni kell, hogy a megoldáshoz használni kell a Pitagorasz tételt, mert ha 45 fokkal elforgatjuk a négyzeteket, akkor négyzet átlója 1,41-el nő. A kirajzolás után 2 másodpercet várjon, mielőtt a futó programot bezárja!	
3. (08_03.py) Készíts programot, amely megrajzolja a mintán lévő ábrát! A jobb oldali alsó „ággal” kell kezdeni! A kis fekete négyzet mindig egyforma, csak a „szára” növekszik 20 kp-al! 10 ága van az alakzatnak, tehát tudni lehet, hogy hány fokkal kell elfordulni mielőtt újabb „ágat” rajzoltatsz ki! A 2 mp itt is kell a végén!	
4. (08_04.py) Készíts programot melyben bekérsz két számot és egy műveleti jelet (karaktert), majd a kiválasztott karakter szerint elvégezzük a műveletet! A programban hozzál létre négy eljárást a műveletek elvégzésére! Figyelj a formátumokra!	
5. (08_05.py) Készíts programot, melyben tízheti lottósorsolást generálsz! Készíts külön eljárást az öt szám generálásához, „generalas()” néven, majd külön eljárást, mellyel kiíratod a tízheti számokat „kiiratas()” néven! A főprogramban csak a kiiratas()-t hívjad meg! A minta szerit formázd a kiíratást!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
6. (08_06.py) Készíts programot, melyben bekér a program egy számot, majd kiírja a négyzetét a számnak! Hozzál létre egy eljárást negyzet(x) néven, melynek a beolvasott számot adjuk értéknek!	
7. (08_07.py) Készíts programot, melyben kiszámítod egy téglatest felszínét és térfogatát! A programban definiálj egy eljárást tegla(a,b,c) néven, mely átveszi a beolvasott számokat! A vonal kirajzolásához is készíts egy vonal() nevű eljárást! A főprogramban csak a két eljárást hívjad meg!	
8. (08_08.py) Készíts programot, melyben a 13f.py és a 13g.py programokhoz hasonlóan zászlót rajzoltatsz ki! A teglalap() eljárás mellet egy bicolor(j,b) nevű eljárást hozzál létre! A zárójelben lévő átadandó értékek a j(obb) és a b(al) szín legyen! A program a lengyel zászlót rajzolja ki a képernyő közepére, a minta szerint! A rajzolószín legyen zöld! A lefutás után legyen 2 mp késleltetés!	
9. (08_09.py) Készíts programot, mely egy 400*300-as tricolor zászlót rajzol ki! A teglalap(), a tricolor(f,k,a) nevű eljárás mellé hozzál létre egy zaszlo(nemzet) nevű eljárást, melyben if-elif-else elágazásban négy fajta zászló legyen megadva! A magyar, osztrák, holland és német. Az else ágban pedig egy („white”, „white”,„white”) színű zászló legyen kirajzolva! A program a képernyő közepére rajzoljon, a minta szerint! A rajzolószín legyen zöld! A lefutás után legyen 2 mp késleltetés!	
10. (08_10.py) Készíts olyan programot, melyben a három fajta zászlót! Kirajzolja a képernyőre a minta szerinti elhelyezésben a lengyel, a magyar és a belga zászlót! A rajzolószín zöld legyen! Az eljárások amelyek szükségesek a feladathoz: • teglalapbi() – a lengyel zászló téglalapja • bicolor(j,b) – a lengyel zászló megrajzolása • teglalaptriv() – a magyar zászló téglalapja • tricolor(f,k,a) – a magyar zászló megrajzolása • teglalapfug() – a belga zászló téglalapja • ftricolor(f, k, a) – a belga zászló megrajzolása	

14. FÜGGVÉNYEK (FUCTION)

A függvények olyan eljárások, amelyek egyetlen értéket visszaadnak a főprogramnak.
Úgy szokták megfogalmazni, hogy a függvénynek visszatérési értéke van.
Legegyszerűbb, ha az Excel függvényekre gondolunk, ott is például egy fkeres függvény alkalmazásakor, több paraméter megadásával a végén egy értéket kapunk, közben tudjuk, hogy mit miért végez el a lépések során. Tehát itt programozáskor olyan, mintha mi terveznénk meg a függvényt, hogy mit kell csinálnia, amely a végén ad egy visszatérési értéket. Az előzőekben már használtunk beépített függvényeket, ilyenek voltak pl. a sqrt(),pow(), abs(), stb.

Nézzünk egyszerű példákat függvény elkészítésére:

(14a.py)

Hozzunk létre egy „plusztiz” nevű függvényt melynek zárójelei közé egy x változót írunk. Ez alapvetően azt jelenti, hogy ha meghívjuk a függvényt, akkor ezt a paramétert vagy argumentumot meg kell adnunk, azaz meg kell mondanunk, mennyi az x értéke. A függvénytörzsbe írjuk be, hogy y = x + 10, a következő sorba pedig azt, hogy return y. Ez a függvény fogja a megadott értéket, és hozzáad 10-et.
Majd végül print utasításban meghívjuk a függvényt 20-as értékkel, igy a program futtatásakor az eredmény 30 lesz.

14a.py

```
1  def plusztiz(x):
2      y=x+10
3      return y
4  print(plusztiz(20))
5
```

Run: 14a

```
E:\00_MM\12_Py
30
```

(14b.py)

A következő programban egy háromszög kerületét számoljuk ki úgy, hogy függvényt készítünk hozzá!
Definiáljunk egy „haromszog_ker” nevű függvényt, melynek a, b, c oldalakat adunk meg bevitelre!
Majd a felhasználótól kérünk be egyész számokat, a minta szerint!
Végül meghívjuk a függvényünket!

14b.py

```
1  def haromszog_ker(a,b,c):
2      print("A háromszög kerülete:",a+b+c)
3
4  a=int(input("Add meg az egyik oldalt: "))
5  b=int(input("Add meg a második oldalt: "))
6  c=int(input("Add meg aharmadik oldalt: "))
7
8  haromszog_ker(a,b,c)
```

Run: 14b

```
E:\00_MM\12_Python_programozas
Add meg az egyik oldalt: 4
Add meg a második oldalt: 6
Add meg aharmadik oldalt: 8
A háromszög kerülete: 18
```

(14c.py)

Ebben a programban kérjünk be két valós számot, melyeket függvény segítségével adunk össze!
Az „osszeg” függvényben is használunk egy return sum parancsot mellyel a visszatérési értéket adjuk meg!

14c.py

```
1  def osszeg(x,y):
2      sum = x + y
3      return sum
4
5  num1 = float(input("Addj meg egy valós számot: "))
6  num2 = float(input("Addj meg még egy valós számot: "))
7
8  print("Az összeg",osszeg(num1,num2))
```

Run: 14c

```
E:\00_MM\12_Python_programozas\progr
Addj meg egy valós számot: 2.6
Addj meg még egy valós számot: 9.4
Az összeg 12.0
```

(14d.py)

Ebben a feladatban tegyük fel, hogy szeretnéd a szobád egyik falát kékre festeni!
Az egyik kérdés az, hogy mennyi festéket kell venned, ha 1 m^2 lefestéséhet 0,15 liter festék kell?
A másik kérdés az, hogy ha 1 liter festék ára 930 Ft, akkor mennyit kell fizetni a szükséges festékért?
A két feladatot különböző módon számoljuk ki.

Először hozzunk létre egy „fest_kalk” nevű függvényt melynek a fal két oldalának méretét kell megadni, és kiszámolja a szükséges festék mennyiségét!
Majd hozzunk létre egy másik függvényt „fest_ar” nevű függvényt, melynek a bemenete szintén a két oldal, a kimenete a kiszámolt ár legyen!

```
14d.py
def fest_kalk(a,b):
    t=a*b
    szuks_fest=t*0.15
    return szuks_fest

def fest_ar(a,b):
    ar=a*b*930
    return ar

a=float(input("Add meg a fal magasságát: "))
b=float(input("Add meg a fal szélességét: "))
liter=fest_kalk(a,b)

print("A szükséges liter: %.2f "% (liter))
print("A festékel ára: %.2f "% fest_ar(a,b))
```

```
Run: 14d
E:\00_MM\12_Python_programozas\pr
Add meg a fal magasságát: 3.6
Add meg a fal szélességét: 5.2
A szükséges liter: 2.81
A festékel ára: 17409.60
```

A főprogramban beolvassuk a két oldal méretét valós számként.
Majd egy „liter” változóba beletesszük a fest_kalk(a,b) függvényt!
Végül kiíratjuk az eredményeket a mintán látható formázással!

(14e.py)

Készítsünk programot függvénnyel, melyben a következő számítást végezzük el:
Az önkormányzat a 20 méternél keskenyebb és 30 méternél rövidebb telkekre 25%-os kedvezményt ad a teljes adóból.
Írjunk függvényt, mely megkapja bemenetként a telek szélességét, hosszát és a teljes adó mértékét! A visszatérési érték a számított adó összege legyen, melyben már figyelembe vesszük a kedvezményt abban az esetben, ha volt!

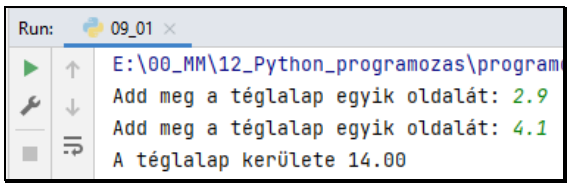
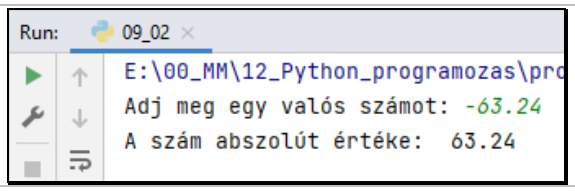
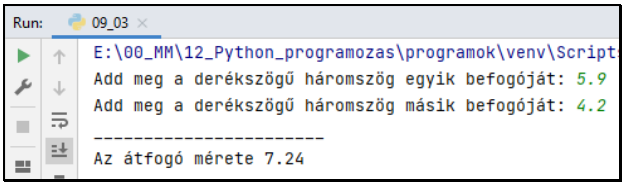
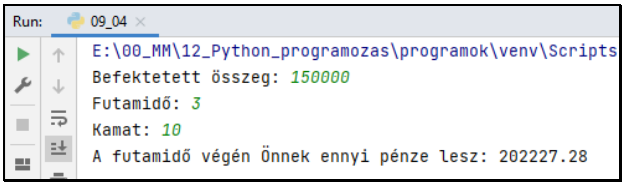
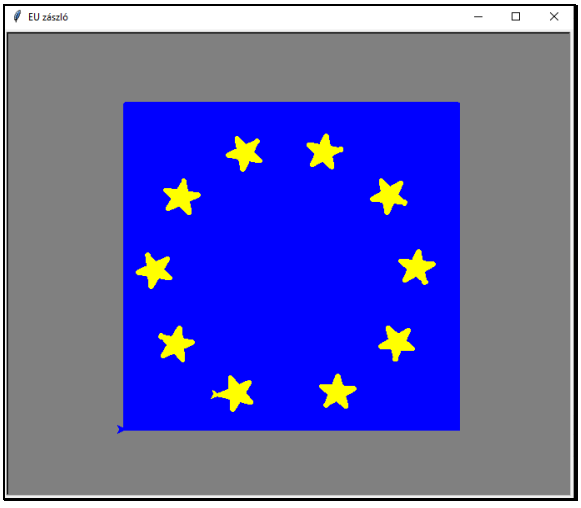
Teszteld a kész programot!

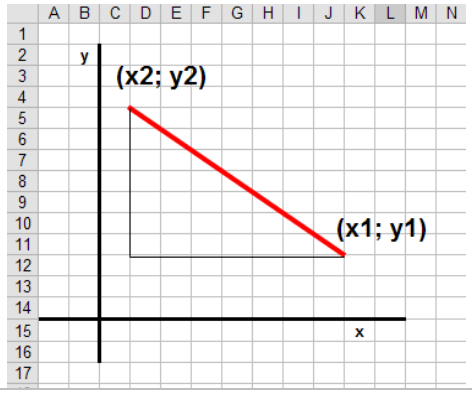
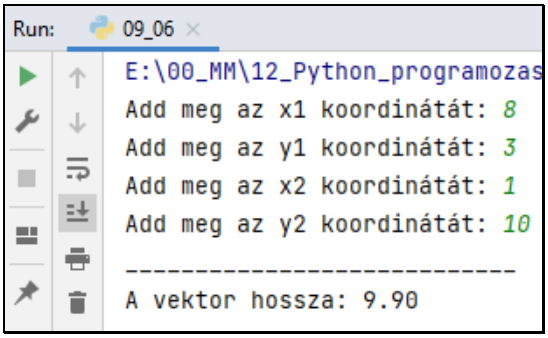
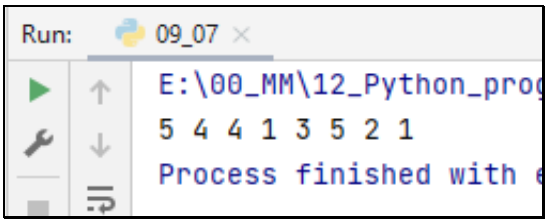
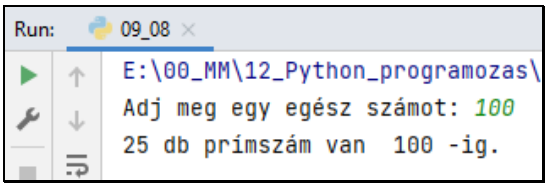
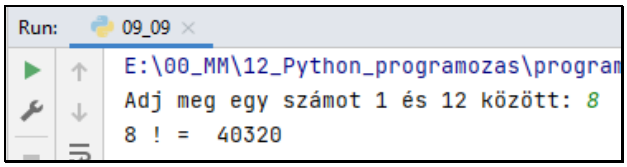
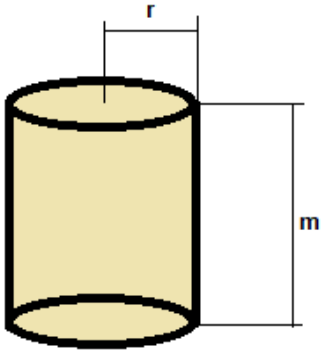
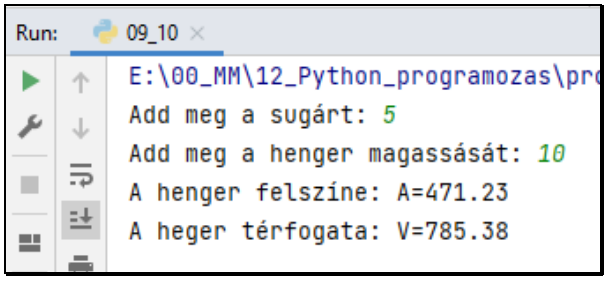
```
14e.py
def kedvezmeny(ado,sz,h):
    if sz<20 and h<30:
        ado=ado*0.75
    return ado
sz=float(input("A telek szélessége: "))
h=float(input("A telek hosszúsága: "))
ado=float(input("A kedvezmény nélküli adó mértéke: "))
print("__________________________")
print("A kedvezményes adó: %.2f" % (kedvezmeny(ado,sz,h)))
```

```
Run: 14e
E:\00_MM\12_Python_programozas\programok\
A telek szélessége: 19.8
A telek hosszúsága: 25.4
A kedvezmény nélküli adó mértéke: 5600
__________________________
A kedvezményes adó: 4200.00
```

```
Run: 14e
E:\00_MM\12_Python_programozas\programok
A telek szélessége: 21.0
A telek hosszúsága: 28.0
A kedvezmény nélküli adó mértéke: 2900
__________________________
A kedvezményes adó: 2900.00
```

GYAKORLATI FELADATOK (14. témakörhöz)

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (09_01.py) Készíts programot, melyben olyan függvényt hozol létre, ahol egy téglalap kerületének kiszámítsa a feladat, úgy hogy a bemenő érték az „a” és a „b” oldal mérete! A visszatérés pedig a kerület értéke! A függvény neve legyen „kerulet”!	
2. (09_02.py) Készíts olyan függvényt, amely visszatér egy bekért valós szám abszolút értékével!	
3. (09_03.py) Készíts programot, melyben egy derékszögű háromszög két befogóját bekéri a program, majd függvény segítségével kiszámolja az átfogó méretét, aztán kiírja a képernyőre a minta szerint az eredményt! Valós számokkal dolgozz!	
4. (09_04.py) Készíts programot, melyben a matematikában sokszor használt „kamatos kamat”-ot számoltatsz függvény segítségével! $$FV = C \cdot \left(1 + \frac{r}{m}\right)^{mt}$$ FV: a kamatos kamat végösszeg C: a befektetett összeg r: éves névleges kamatláb (kamat) m: évközi kamatfizetés száma (ebben az esetbe 12) t: évek száma (futamidő)	
5. (09_05.py) Készíts programot, amely az EU zászlót rajzolja ki a képernyőre! Az ablak címsorában az „EU zászló” felirat jelenjen meg! Az ablak háttérszíne legyen szürke! Két teknős legyen definiálva! Az egyik neve legyen „teki_1”, kék rajzoló és kitöltő színnel! A másik neve legyen „teki_2”, sárga rajzolószínnel és sárga kitöltőszínnel! A kék zászló alap legyen pozícionálva: kiindulópontja (-200,-200), mérete legyen (400*400)! A csillag vonalai legyenek 40 kp-ok! 10 db csillagot rajzoljon ki! Alkalmazz függvényeket!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
6. (09_06.py) Írj programot, melyben létrehozol egy függvényt, ami visszaadja egy vektor hosszát! $a^2+b^2=c^2$ 	
7. (09_07.py) Készíts programot, melyben véletlenszerűen 1 és 10 darabszám között generál kockadobásokat! Majd egymás mellé kiírja a generált számokat. Programot úgy készítsd el, hogy külön függvényben generálsz 1-6.ig véletlen számokat [kockadobas()], majd eszt egy másik függvényben felhasználod a kiíráshoz, ahol a kiírás számát 1-10-ig szintén véletlenszerűen generálod [darab()].	
8. (09_08.py) Készíts programot, függvények létrehozásával, melyben bekér egy számot, majd megszámolja, hogy hány darab prímszám van addig a számig!	
9. (09_09.py) Készíts programot függvények létrehozásával, melyben bekérsz egy számot 1 és 12 között, majd kiszámolja a program a szám faktoriálisát!	
10. (09_10.py) Készíts programot, függvények létrehozásával, melyben kiszámolod egy henger felszínét és térfogatát! 	

15. LISTÁK

Vannak olyan esetek, amikor sok (pl.: 50 db) változóra van szükségünk. Ebben az esetben sok időbe telne a az értékadás és nehézkes lenne a nevek megjegyzése. Ezért megoldást jelent, a sok változónak egyetlen nevet adunk, de mellette sorszámozzuk őket. Például (hónapok(4), napok(6), stb.

A lista névvel és sorszámokkal ellátott összetett változó. A lista elemeire nevével és a sorszámukkal hivatkozunk.

A listaelem sorszámát **indexnek** nevezzük.

Arra kell figyelni, hogy a **sorszámozás nulla értékkel indul**. (Pl.: január -> 0)

Tehát: honap(2):= „március”

A 12 elemű lista utolsó tömbindexe a 11.

A lista egy sokoldalú adatszerkezet, műveleteket hajthatunk végre rajtuk; törölhetünk benne, rendezhetjük az értékeket, stb.

Fontos, hogy a kör alakú zárójelek helyett négyzetes zárójeleket kell használni!

pl.: honapok=[„Január”,”Február”,…,”December”]

0	Január
1	Február
2	Március
3	Április
4	Május
5	Június
6	Július
7	Augusztus
8	Szeptember
9	Október
10	November
11	December

(15a.py)

Az első listás feladat nagyon egyszerű. Hozzunk létre egy „honap” nevű listát, melyet feltöltünk a hónapok neveivel! Majd kiíratjuk a lista hatos számú, pontosabban a hetedik elemét a júliust!

```
honapok=["Január","Február","Március","Április","Május","Június",
         "Július","Augusztus","Szeptember","Október","November","December"]
print(honapok[6])
```

```
Run: 15a
C:\Users\kolma
Július
```

(15b.py)

Mentsd el másként az előző programot, és változtasd meg úgy, hogy írja ki a hónapok neveit külön sorokba a mint szerint!

Ehhez használjuk a „for” utasítást!

```
honapok=["Január","Február","Március","Április","Május","Június",
         "Július","Augusztus","Szeptember","Október","November","December"]
for i in range(12):
    print(honapok[i])
```

```
Run: 15b
C:\Users\ko
Január
Február
Március
Április
Május
Június
Július
Augusztus
Szeptember
Október
November
December
```

Tehát többféle módon lehetséges egy **új lista létrehozása**; legegyszerűbb az elemek szögletes zárójelbe való felsorolása ([és]):

szamok = [10, 20, 30, 40]

gyumolcsok = ["alma", "eper", "barack", ”körte”]

Az első példa egy lista, amely négy egész számot tartalmaz. A második lista pedig három sztringet tartalmaz. A lista elemeinek nem kell azonos típusúnak lennie. A következő lista tartalmaz egy szrtinget, egy valós számot, egy egész számot és (érdekességképpen) egy másik listát.

vegyes_lista = ["hello", 2.0, 5, [10, 20]]

A listában szereplő másik listáról azt mondjuk, hogy beágyazott lista.

Végül azt a listát, amely nem tartalmaz elemeket, üres listának nevezzük, és [] jelöljük.

(15c.py)

A lista hosszának használata a már tanult len() függvénnyel lehetséges.
Tehát ha nem tudjuk konkrétan hogy a lista hossza mekkora, akkor a len() függvény megmondja.
Nézzünk példát erre!
Soroljuk fel a sakkfigurák neveit egy listába, majd for ciklussal „i”-től menjünk a listán végig! Használjuk a len() függvényt a mint szerint!

15c.py

```
1  sakk_figurak=["Királynő","Király","Bástya","Futó","Ló","Gyalog"]
2
3  for i in range(len(sakk_figurak)):
4      print(sakk_figurak[i])
```

Run: 15c

```
E:\00_MM\
Királynő
Király
Bástya
Futó
Ló
Gyalog
```

(15d.py)

Ebben a példában létrehozunk egy olyan listát, amelyben beágyazások is találhatók. De amikor kiíratjuk a lista hosszát, akkor csak a ”fő” lista darabszámát írja ki!

15d.py

```
1  lista_1=["sakkfigurák",8,["Gyalog","Futó","Király"],[1,2,3],2.5]
2  hossz = len(lista_1)
3  print(hossz)
```

Run: 15d

```
E:\0
5
```

(15e.py)

Listaműveletek:

listák összefűzése „+” operátorral

listák többszörözése „*” operátorral

15e.py

```
1  a = [1, 2, 3]
2  b = [4, 5, 6]
3  c = a + b
4  print(c)
5  print("----------------------")
6  d = [0] * 4
7  print(d)
8  e = [1, 2, 3] * 3
9  print(e)
10 print("----------------------")
```

Run: 15e

```
C:\Users\kolmank\AppData\Loca
[1, 2, 3, 4, 5, 6]
----------------------
[0, 0, 0, 0]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
----------------------
```

(15f.py)

Listák szeletelése

Listák módosítása

Új elem beszúrása adott helyre

Listában lévő tagok törlése

15f.py

```
1  sakk_figurak=["Királynő","Király","Bástya",
2                "Futó","Ló","Gyalog"]
3  print(sakk_figurak[1:3])
4  print(sakk_figurak[:4])
5  print(sakk_figurak[3:])
6  print(sakk_figurak[:])
7
8  print("----------------------")
9  sakk_figurak=["Király","Királynő","Bástya
10               "Futó","Ló","Gyalog"]
11 sakk_figurak[1]="Vezér"
12 sakk_figurak[4]="Huszár"
13 print(sakk_figurak)
14
15 print("----------------------")
16 sakk_figurak=["Királynő","Futó","Ló","Gyalog"]
17 sakk_figurak[1:1]="Király","Bástya"
18 print(sakk_figurak)
19
20 print("----------------------")
21 del(sakk_figurak[2:4])
22 print(sakk_figurak)
```

Run: 15f

```
C:\Users\kolmank\AppData\Local\Programs\Python\Python38\py
['Király', 'Bástya']
['Királynő', 'Király', 'Bástya', 'Futó']
['Futó', 'Ló', 'Gyalog']
['Királynő', 'Király', 'Bástya', 'Futó', 'Ló', 'Gyalog']
----------------------
['Király', 'Vezér', 'Bástya', 'Futó', 'Huszár', 'Gyalog']
----------------------
['Királynő', 'Király', 'Bástya', 'Futó', 'Ló', 'Gyalog']
----------------------
['Királynő', 'Király', 'Ló', 'Gyalog']
```

Ebben a feladatban a mintaprogram alapján értelmezzük a kódokat, hogy mi miért történik!

(15g.py)

Lista metódusok

A „pont” operátor is használható a lista objektumok beépített metódusainak elérésére. Próbáljuk ki a leghasznosabb metódusokat, amelyekkel módosíthatunk a listáinkon!

- Elöször létrehozunk egy üres listát, majd feltöltjük elemekkel! Ehhez az **új_lista.append(x)** utasítást használjuk a minta alapján
- Beszúrunk 5 db számot a listánkba egymás után sorban.
- Majd mindig kiíratjuk a lista tartalmát a képernyőre, hogy lássuk a változást!
- A második helyre (a 9-es helyére) beszúrunk egy 15-ös értéket, az **uj_lista.insert(2,15)** utasítással!
- Az **uj_lista.count(15)** utasítással megszámolhatjuk, hogy hányszor fordul elő a listánkban a 15-ös szám!
- Az eddigi listánk végére hozzáadhatunk egy másik listát a **uj_lista.extend([5, 9, 5, 11])** utasítással!
- Keressük megy az index értékét, sorszámát a listában az **print(uj_lista.index(9))** utasítással!
- Fordítsuk meg a listánkat az **uj_lista.reverse()** utasítással!
- Rendezzük növekvő sorrendbe a listánk értékeit az **uj_lista.sort()** utasítással!
- Töröljük, távolítsuk el az első előfordulását a 15-ös számnak az **uj_lista.remove(15)** utasítással!

Gépeljük be a kódot, kövessük végig a parancsokat, és a megjelenített eredményt!

```
15g.py
uj_lista = []
uj_lista.append(15)
uj_lista.append(27)
uj_lista.append(9)
uj_lista.append(12)
uj_lista.append(23)
print(uj_lista)
print("_________________")
uj_lista.insert(2, 15)
print(uj_lista)
print("_________________")
print(uj_lista.count(15))
print("_________________")
uj_lista.extend([5, 9, 5, 11])
print(uj_lista)
print("_________________")
print(uj_lista.index(9))
print("_________________")
uj_lista.reverse()
print(uj_lista)
print("_________________")
uj_lista.sort()
print(uj_lista)
print("_________________")
uj_lista.remove(15)
print(uj_lista)
```

```
Run: 15g
C:\Users\kolmank\AppData\Local\Program
[15, 27, 9, 12, 23]
-----------------
[15, 27, 15, 9, 12, 23]
-----------------
2
-----------------
[15, 27, 15, 9, 12, 23, 5, 9, 5, 11]
-----------------
3
-----------------
[11, 5, 9, 5, 23, 12, 9, 15, 27, 15]
-----------------
[5, 5, 9, 9, 11, 12, 15, 15, 23, 27]
-----------------
[5, 5, 9, 9, 11, 12, 15, 23, 27]
```

(15h.py)

```
15h.py
szamok=[12,6,8,11,5,7,9,2,1]
print("A számok lista tartalma: ",szamok)
masolat_1=szamok.copy()
print("A másolat lista tartalma: ",masolat_1)
szamok_2 = szamok
print("A számok 2 lista tartalma: ",szamok_2)
szamok.remove(12)
szamok.remove(11)
print("A számok lista tartalma: ",szamok)
print("A másolat lista tartalma: ",masolat_1)
print("A számok 2 lista tartalma: ",szamok_2)
```

Ebben a példában egy fix listáról készítünk **másolat**ot, majd egy **hivatkozás**t! Aztán az eredeti listából törlünk két elemet. Látszik, hogy **a hivatkozásból szintén eltűntek az elemek, a másolatból nem**.

```
Run: 15h
E:\00_MM\12_Python_programozas\programok\venv\Scripts\pyth
A számok lista tartalma:  [12, 6, 8, 11, 5, 7, 9, 2, 1]
A másolat lista tartalma:  [12, 6, 8, 11, 5, 7, 9, 2, 1]
A számok 2 lista tartalma:  [12, 6, 8, 11, 5, 7, 9, 2, 1]
A számok lista tartalma:  [6, 8, 5, 7, 9, 2, 1]
A másolat lista tartalma:  [12, 6, 8, 11, 5, 7, 9, 2, 1]
A számok 2 lista tartalma:  [6, 8, 5, 7, 9, 2, 1]
```

(15i.py)

Lista bejárása

for ciklussal

Értelmezd a példákat!

```
15i.py
napok =["Hétfő","Kedd","Szerda","Csütörtök"
       "Péntek","Szombat","Vasárnap"]
for i in napok:
    print(i, end=" ")
print()
for j in range(len(napok)):
    print("%d. nap: %s" % (j+1, napok[j]))
```

```
Run: 15i
E:\00_MM\12_Python_programozas\programok\venv\Script
Hétfő Kedd Szerda Csütörtök Péntek Szombat Vasárnap
1. nap: Hétfő
2. nap: Kedd
3. nap: Szerda
4. nap: Csütörtök
5. nap: Péntek
6. nap: Szombat
7. nap: Vasárnap
```

(15j.py)

Karakterláncok darabolása listává

Gyakran az összetartozó adatokat egymás mellett szóközzel, vesszővel, pontosvesszővel választjuk el egymástól. Adatbázisokban ezeket hívjuk rekordoknak. Például ha egy autó adatait nézzünk:
AAA-111, szűrke, Citroen, C4, 2012, 3500000
Mindenki érti, hogy melyik adat mit jelenthet. Ezekre az adatokra általában külön lehet szükségünk, ezért nézzünk pár parancsot, melyekkel változtatunk a struktúrán! (Tulajdonképpen feldolgozhatóvá tesszük az adatokat!)
Tömböt készítünk, melynek elemeivel műveleteket végezhetünk!

```
16c.py
1  auto="AAA-111, szűrke, Citroen, C4, 2012, 3500000"
2  auto=auto.replace(",","")
3  print(auto)
4  print("_____________________")
5  auto_adatok=auto.split()
6  print(auto_adatok)
7  print("_____________________")
8  for i in auto_adatok:
9      print(i)
```

```
Run: 16c
C:\Users\eloado\PycharmProjects\pythonProject\venv\Scripts
AAA-111 szűrke Citroen C4 2012 3500000
_____________________
['AAA-111', 'szűrke', 'Citroen', 'C4', '2012', '3500000']
_____________________
AAA-111
szűrke
Citroen
C4
2012
3500000
```

- Az első sorban megadunk egy sztring adatsort vesszőkkel elválasztva.
- A második sorba, az „auto” változóba visszatesszük a vesszők nélküli szöveget. Erre a **replace()** utasítást használjuk a minta szerint. A vesszők helyére a („”) semmit tesszük. Erre azért van szükség, mert a darabolásnál, ha bent marad a vessző, akkor az a szó végén maradna egy felesleges vessző karakter. (pl.: fehér,).
- Aztán ellenőrzésül kiíratjuk az aktuális auto változó tartalmát. Vonallal elválasztva a következő soroktól. (3-4. sor)
- Majd behozunk egy új tömböt (auto_adatok néven), mely tömbbe a szóközök mentén feldarabolt szöveget beleteszem egyesével. Erre a **split()** parancsot használjuk a minta alapján. Így lesznek a különböző adat elemek egy tömb elemei. (5. sor)
- Aztán kiíratom egy sorba a tömböt, hogy lássuk az eredményt. (6. sor)
- Végül a tömb elemeit kiíratom egymás alá for utasítás segítségével, hogy lássuk hogy az elemekkel külön -külön dolgozhatunk. (8-9. sor)

(15k.py)

Képzeljünk el egy autókölcsönzőt! Az előző programból kiindulva most több autó (4db) adatait tároljuk egy tömbben.

Készítsünk programot, melynek az eredménye a jobb oldali képernyőkép!

A kérdés az, hogy átlagosan hány évesek az adatbázisban lévő autók?
Mennyi a legdrágább autó ára?

```
Run: 16d
C:\Users\kolmank\AppData\Local\Programs\Python\Python38\pyt
['AAA-111 szűrke Citroen C4 2012 3500000', 'BBB-222 fehér A
_____________________
['AAA-111', 'szűrke', 'Citroen', 'C4', '2012', '3500000']
['BBB-222', 'fehér', 'Audi', 'A4', '2015', '5500000']
['CCC-333', 'kék', 'BNW', 'Q6', '2020', '25000000']
['DDD-444', 'zöld', 'Skoda', 'Octávia', '2008', '1300000']
_____________________
Az autók átlagéletkora:  8.25  év
A legdrágább autó ára:  25000000  Ft.
```

Nézzük részletesen a programunkat:

```
16d.py
1  import datetime
2  auto=["AAA-111, szűrke, Citroen, C4, 2012, 3500000",
3        "BBB-222, fehér, Audi, A4, 2015, 5500000",
4        "CCC-333, kék, BNW, Q6, 2020, 25000000",
5        "DDD-444, zöld, Skoda, Octávia, 2008, 1300000"]
6  ma = datetime.date.today()
7  #print(ma.year)
8  for i in range(4):
9          auto[i]=auto[i].replace(",","")
10 print(auto)
11 print("----------------------")
12 evek=0
13 atlag=0
14 max=0
15 for i in range(4):
16       auto_adatok=auto[i].split()
17       print(auto_adatok)
18       if max<int(auto_adatok[5]):
19             max=int(auto_adatok[5])
20       evek=evek+int(auto_adatok[4])
21       atlag=evek/4
22 print("----------------------")
23 print("Az autók átlagéletkora: ",ma.year-atlag," év")
24 print("A legdrágább autó ára: ",max," Ft.")
```

- A programunk első sorában meghívjuk a „datetime” modult, mert a későbbiekben szükségünk lesz az aktuális dátumra.
- A második sortól létrehozzuk az auto nevű tömböt, melybe begépeljük a négy autó adatait vesszőkkel elválasztva, a minta szerint!

AAA-111, szűrke, Citroen, C4, 2012, 3500000
BBB-222, fehér, Audi, A4, 2015, 5500000
CCC-333, kék, BNW, Q6, 2020, 25000000
DDD-444, zöld, Skoda, Octávia, 2008, 1300000

- A hatodik sorban egy ma nevű változóba beletesszük az aktuális dátumot!
  ma=datetime.date.tuday()
  Majd a következő sorba kiíratjuk a képernyőre! Később ezt # karakterrel megjegyzésbe tesszük! Sőt mivel az autók átlagéletkorának kiszámításához csak az évre lesz szükség, a dátumról az évet jelenítjük csak meg! (*ma.year()*)
- A nyolcadik, kilencedik sorban kivesszük a vesszőket a szövegünkből úgy, hogy a négy tömbelemen végig megyünk for utasítással a replace() parancs felhasználásával!
- Ellenőrzésképpen kiíratjuk az auto tömb tartalmát és beszúrunk egy választóvonalat a következő sorba!
- A 12-14 sorokban létrehozunk változókat, melyekre szükség lesz a számolásokban. Lenullázzuk őket!
- A 15. sortól indul a program lényege.
  Ugye szükségünk van az autók átlag életkorára és a legdrágább autó árára!
  Ehhez az adatink közül a rendszámra, színre, márkára, típusra nincs szükség.
  Viszont a tömbünk elemeinek ötödik „elemére” az évszámra igen, és a legvégén szereplő milliós összegre is!
  Arra kell figyelni, hogy ezek az értékek stringek, és arra is, hogy a belső tömbünk ötödik és hatodik (a 0 kezdősorszám miatt 4. és 5.) helyen állnak!
- Először for ciklust indítok, hogy végig menjen a 4 autó adatán!
- Aztán a split() utasítással a szóközök mentén újabb tömböket alakítok ki, auto_adatok néven! (14. sor)
- Kiíratom a tömb elemeit, soronként! (15. sor)
- Aztán megvizsgálom, hogy a max aktuális értéke nagyobb-e mint az auto_adatok utolsó eleme (az autó ára)! Mert ha igen akkor cseréljük, különben megy tovább a program!
- Majd az autók átlagéletkorának kiszámításához összeadom az autók gyártási éveit, majd elosztjuk néggyel. Erre kapunk egy 2013.75-ös értéket!
- Végül kiíratjuk az eredményeket a minta szerint!
- Az átlagéletkort úgy kell kiíratni, hogy mindig az aktuális évhez képest mondja meg a program, hogy mennyi idősek az autók. Ezért akkor fog pontosan működni a programunk, ha a „ma” változó évéből kivonjuk az autók gyártási évének átlagát!
- Futtassuk a programunkat, és javítsuk az esetleges hibákat! Mindig úgy használjuk a különböző utasításokat, hogy végiggondoljuk, hogy mi miért történik, és tudatosan haladjuk a sorokban! Ne csak másolás legyen!

Listák leképezése (list comrehension)

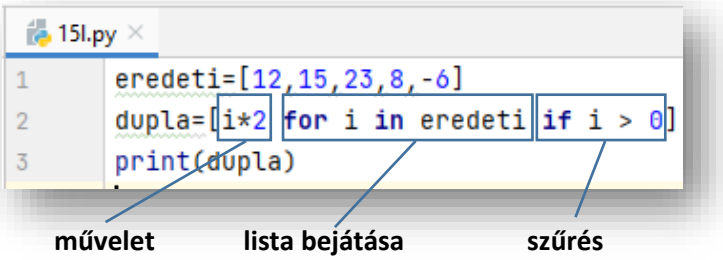
Ez a kifejezés egy kicsit idegenül hangozhat számunkra, de ha egyszerűbben akarunk fogalmazni, akkor mondhatjuk hogy a **listákat feldolgozzuk azokkal műveleteket végzünk**. **Tulajdonképpen lerövidíthetjük a kódunkat**. **Összevonhatjuk a műveleteket egy sorba!**
Nézzünk egy egyszerű példát:

- Vegyünk egy „szamok” nevű öt elemű listát **-> szamok=[12,15,23,8,-6]**
- A feladat az, hogy a lista elemeinek értékét duplázzuk meg, akkor, ha a szám pozitív és az eredményül kapott értékeket tároljuk egy „dupla” nevű listában. **-> dupla =[24,30,26,16]**
- Ezt meg tudjuk valósítani a mostani tudásunk alapján is a következők szerint:

```
1    szamok = [12,15,23,8,-6]
2    dupla = []
3    for i in szamok:
4        if i > 0:
5            dupla.append(i * 2)
6    print(dupla)
```

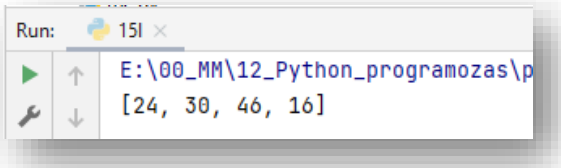
Nézzük meg ennek a feladatnak az egyszerűbb megoldását a leképezést, amikor összevonjuk a műveleteket egy sorba:

(15l.py)



Tehát az „eredeti” nevű listában szereplő számok közül duplázzuk meg a pozitív tagok értékét és tegyük bele egy „dupla” nevű új listába, majd írassuk ki a képernyőre a „dupla” lista tartalmát!

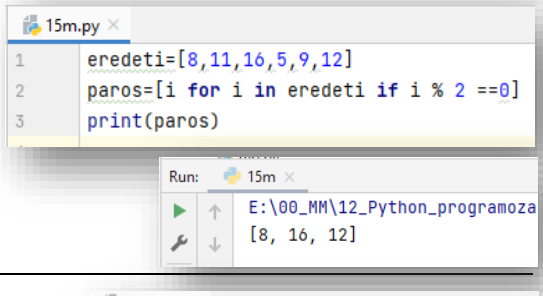
- Először töltsük fel az „eredeti” listát a minta szerinti számokkal.
- Majd a második **sor elején megadjuk az új lista nevét, és az egyenlőség jel után a két szögletes zárójelet!**
- **A zárójelen belül először mindig a lista bejárását adjuk meg az eddig tanult módon.** (for i in eredeti)
- **Majd megadjuk az elejére, hogy milyen műveletet szeretnénk elvégezi.** (i*2)
- **Végül az esetleges feltételt is megadjuk a művelet bejárása után.** (if i>0)
- Aztán a harmadik sorban egyszerűen kiíratjuk a „dupla” nevű lista tartalmát.



(15m.py)

A feladatban gyűjtsük ki a páros számokat egy új listába!

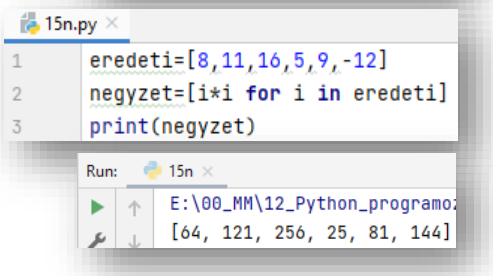
- A lista bejárása és a szűrés egyértelmű!
- A művelet megadásánál csak egyszerűen beletesszük a páros nevű listába. Tehát csak egy „i”-t adunk meg.



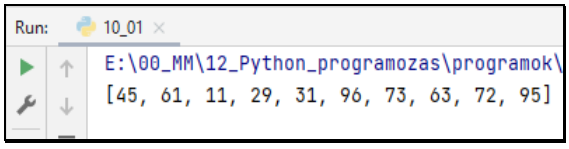
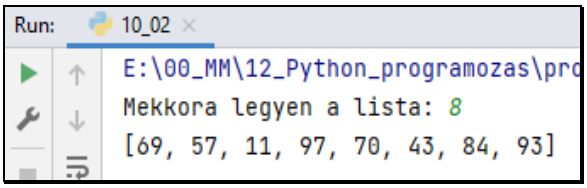
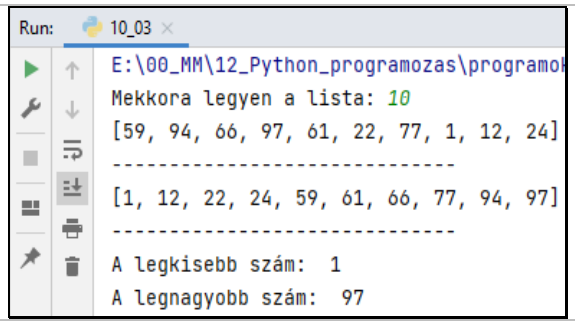
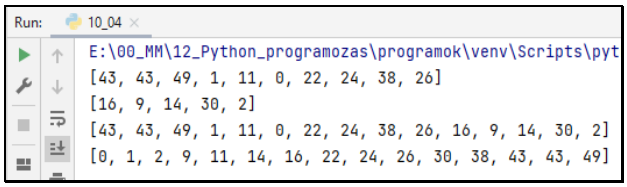
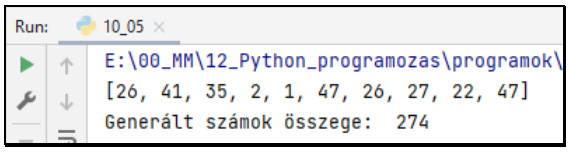
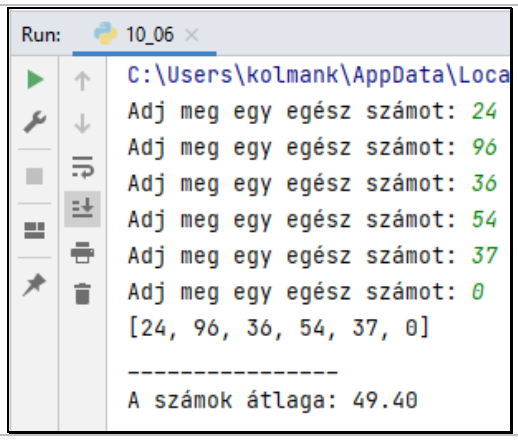
(15n.py)

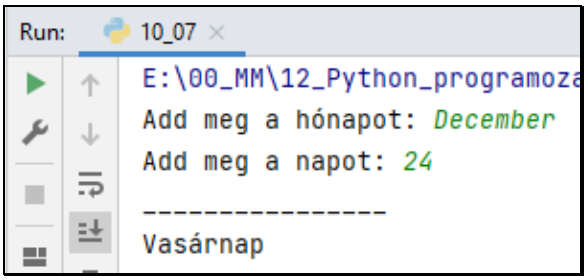
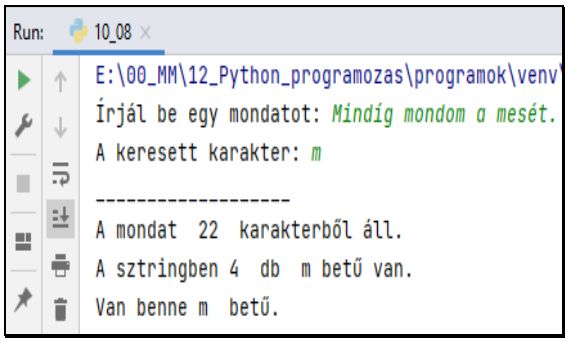
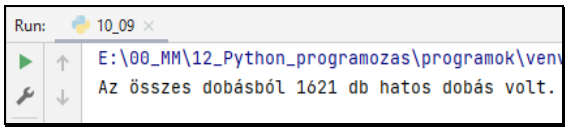
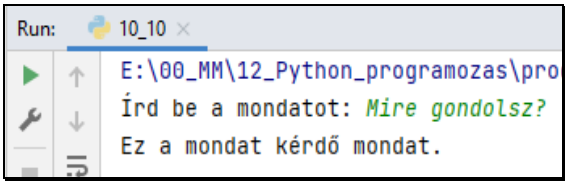
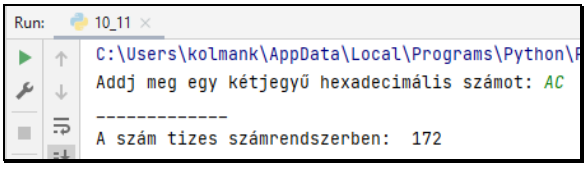
A feladatban az „eredeti” listában lévő számokat kell négyzetre emelni és egy új listába beletenni.

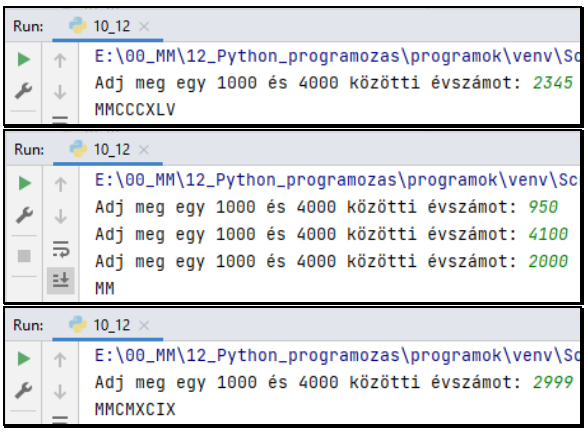
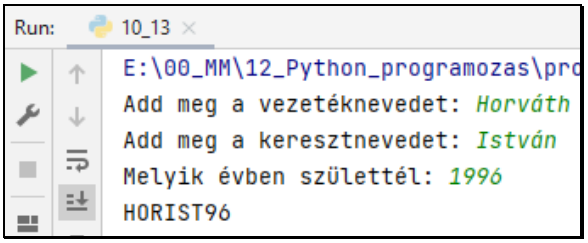
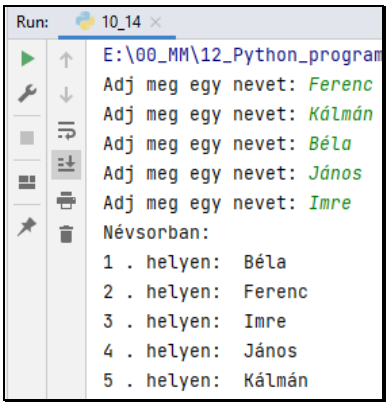
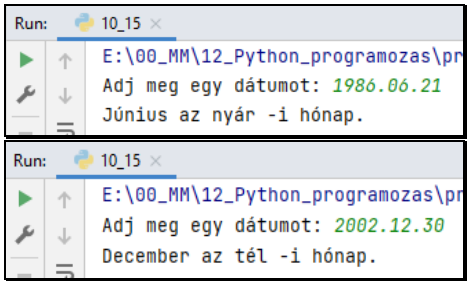
- Ha nincs feltétel egyszerűen kihagyjuk!



GYAKORLATI FELADATOK (15. témakörhöz)

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (10_01.py) Készíts programot, mellyel feltöltesz egy 10 elemű listát véletlen 1 és 100 közötti egész számokkal!	
2. (10_02.py) Mentsd el az előző programot másként, és változatasd meg a következőképpen: A program elején az üres lista neve egyen: szamok A feltöltést függvénnyel old meg! A függvény neve legyen: feltölt()! Az, hogy mekkora legyen a lista mérete azt a felhasználó adja meg!	
3. (10_03.py) Mentsd el az előző programot másként, és egészítsed ki a következőképpen: A minta szerint először rendezd növekvő sorba a listát! Majd írasd ki a legkisebb és a legnagyobb számot!	
4. (10_04.py) Készíts egy programot, melyben függvénnyel létrehozol egy 10 (lista_a) és egy 5 elemű (lista_b) tömböt! Majd egy harmadik tömbben (lista_c) összeadod a két tömb tartalmát! Végül rendezed a harmadik tömböt a minta szerint!	
5. (10_05.py) Készíts programot, melyben függvénnyel generálsz egy 10 elemű listába véletlen számokat 1 és 100 között! Majd a lista elemeit add össze és kiírasd ki a képernyőre a minta szerint!	
6. (10_06.py) Készíts programot, amely nulla végjelig bekér egész számokat, amiket listába helyez! Majd a végjel után kiírja a listát és a beírt számok átlagát a minta szerint! A nulla végjelet ne számolja bele az átlagba!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
7. (10_07.py) Készíts programot, amely megmondja, hogy ha január 1. hétfőre esik, akkor a felhasználó által begépelt hónap, nap, milyen napra esik! Ehhez a feladathoz segítséget kapsz. • Kell 3 lista: ○ honapok[január,…,december] ○ napok[hétfő,..,vasárnap] ○ *napszam=[0,31,59,90,120,151,181,212 ,243,273,304,335]* • bekéred a hónapot és a napot (ho,nap) • a következő képlet kiszámolja a nap sorszámát • *napsorszam=(napszam[honapok.index(ho)-1]+nap) % 7* • *hetnapja=napok[napsorszam]* • majd a vonal után kiíratjuk a hét napját	
8. (10_08.py) A begépelt mondatokat is kezelhetjük tömbként. Ugyanúgy, mehetünk végig karakterenként pl. „for” ciklussal. Készíts programot, melyben a minta szerint bekérsz egy mondatot, amit egy tömbben tárolsz! Majd a keresendő karaktert kérje be a program! A minta szerint egy vonal után írasd ki, hogy hány karakterből áll a mondat! Aztán írasd ki, hogy hány darab van a keresett karakterekből a mondatban! Ennél a for-in utasítást alkalmazd! Végül írasd ki, hogy van, illetve nincs benne a keresett karakter. Ezt a feladatot az if-in-else utasítással old meg	
9. (10_09.py) Készíts programot, melyben egy tömbben generálsz 10000 darab kockadobást! Majd megszámoltatod, hogy abból hány darab volt 6-os!	
10. (10_10.py) Készíts egy programot, amely a felhasználó által megadott mondatról a mondatvégi jel alapján eldönti milyen fajtájú! (kijelentő, kérdő, felkiáltó / felszólító / óhajtó)	
11. (10_11.py) Készíts programot, mely átvált egy kétjegyű hexadecimális számot decimális számmá! A feladat megoldásához használj listát!	

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
12. (10_12.py) Készíts egy programot, amely egy 1000 és 4000 közötti decimális arab számot átvált római számmá! Készíts listákat a római számokkal! (egyesek, tízesek, százasok ezresek) Csak az intervallumon belüli számokat fogadja el, ha nem felel meg, akkor kérjen be újat! A program megírása során figyelj arra, hogy a kerek számok is működjenek! Tesztelj! A római számot szorosan egymás után írasd ki a minta szerint! (A római számoknál nincs 0-s (helyi)érték leírva ezért használj a listában a nulladik helyen („”) üres karaktert! (Kivéve az ezresnél!))	
13. (10_13.py) Készíts programot, mely egy bekért vezetéknévből, keresztnévből, és egy születési évből készít egy felhasználónevet! A karakterek legyenek nagybetűsek! A felhasználónév kialakítása úgy történjen, hogy a nevek első három karakteréből és az évszám utolsó két számából alakítja ki a minta szerint!	
14. (10_14.py) Készíts programot, melyben egy „nevek” nevű tömbben tárolsz öt keresztnevet, melyeket azután kiíratsz névsorban a minta szerint!	
15. (10_15.py) Készítsél programot, amely bekér egy dátumot! Majd kiíratod, hogy melyik hónap van a dátumban és az melyik évszakban van! A feladatot két lista létrehozásával készítsd el (honap, evszak)!	

16. TÖBBDIMENZIÓS LISTA – MÁTRIX, TÖMB

```
matrix=[['A','B','C'],['D','E','F'],['G','H','I']]
matrix=[['A','B','C'],
        ['D','E','F'],
        ['G','H','I']]
```

Ahogy az előző fejeztben is említettük a listánk egyik eleme lehet egy másik lista. Sőt egy lista minden eleme is lehet (belső) lista. Ilyenkor beszélünk többdimenziós listásól.
Tehát a „mátrix” lista első (0.) eleme: =['A','B','C']
és annak az első (0.) eleme: 'A'
Mindig a nagyobb (külső) listából indulunk a kisebb (belső) lista felé!
A többdimenziós listákat mindig téglalapként képzeljük el, írjuk fel! (második példa)

Az elemek téglalapszerű elrendezését kétdimenziós listának, vagy mátrixnak nevezzük.
Az első index a sort (sorindex), a második az oszlopot (oszlopindex) jelöli.
Azt nem szabad elfelejteni, hogy itt is az index értéke a nullától indul.

0,0	0,1	0,2
1,0	1,1,	1,2
2,0	2,1	2,2

A	B	C
D	E	F
G	H	I

Tehát a matrix[1][2] ➔ F
Nem kell azonos sor és oszlopszámúnak lennie a tömböknek!
pl. n=5, m=6 (5*6)

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]	[4][5]

Mondatszerű leírásban a tömbök megadása pl.: matrix(2,4)
Pythonban az előbb említett módon **pl.: matrix[2][4]**
Többdimenziós tömbök készítésénél fokozottan figyeljünk a négyzetes zárójelek „[]” és a vesszök”,” helyes használatára!

(16a.py)

Készítsünk programot, melyben megadunk és kiíratunk egy 3*3-as tömböt! A feladatban egy „amőba” játék tábláját jelenítsük meg!
A táblában az üres helyet „_” karakter, a másik kettőt pedig a „X” és „O” karakterek jelenítsék meg!

```
16a.py
tabla=[['_','X','0'],
       ['X','_','0'],
       ['_','0','X']]
for i in range(0,3):
    for j in range(0,3):
        print(tabla[i][j],end=" ")
    print()
```

- Az első három sorban megadjuk a tábla elemeit. Sorokba tördelve megfelelő struktúrával! Az elemek megfelelő elválasztására figyeljünk!
- Majd indítunk egy külső ciklust, melyben a sorokat fogjuk váltani. (0,1,2 sorok)
- Aztán egy belső ciklust indítunk, amelyben a sorokban lévő elemeit íratjuk ki a minta szerint. A karaktereket szóközökkel választjuk el egymástól!
- Majd az utolsó sorban a külső ciklus sortörése miatt egy print() utasítást gépelünk be. Új sort kezdünk.
- Futtassuk és teszteljük a programunkat!

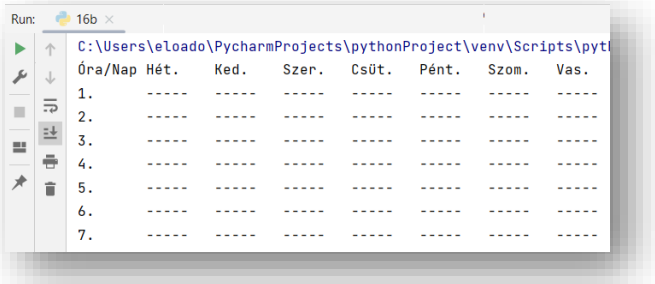
```
Run: 16a
E:\00_MM\12_Python_pr
_ X 0
X _ 0
_ 0 X

Process finished with
```

(16b.py)

Készítsünk programot, amely az előző példához hasonlóan egy fix tömböt kiírat a képernyőre!
Ebben a példában egy órarendet szeretnénk kiíratni a képernyőre, soronként soron belül tabulátorokkal elválasztva.

```
16b.py
orarend=[['Óra/Nap','Hét.','Ked.','Szer.','Csüt.','Pént.','Szom.','Vas.'],
         ['1.  ','-----','-----','-----','-----','-----','-----','-----'],
         ['2.  ','-----','-----','-----','-----','-----','-----','-----'],
         ['3.  ','-----','-----','-----','-----','-----','-----','-----'],
         ['4.  ','-----','-----','-----','-----','-----','-----','-----'],
         ['5.  ','-----','-----','-----','-----','-----','-----','-----'],
         ['6.  ','-----','-----','-----','-----','-----','-----','-----'],
         ['7.  ','-----','-----','-----','-----','-----','-----','-----']]
for i in range(0,8):
    for j in range(0,8):
        print(orarend[i][j],end="\t")
    print()
```

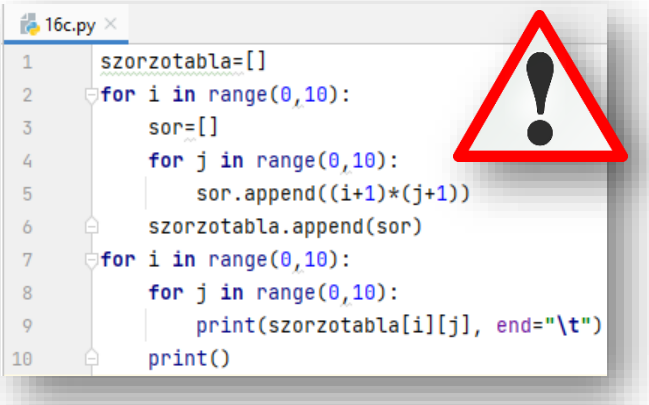
- A program elején megadjuk egy 8*8-as tömbben az első sorban a napokat, majd a sorszámokat és az órák helyét 5db „-„ karakterrel.
- Majd egymásba ágyazott ciklussal 0-7-ig kiíratjuk a tömb elemeit tabulátorokkal elválasztva, majd sort váltunk.

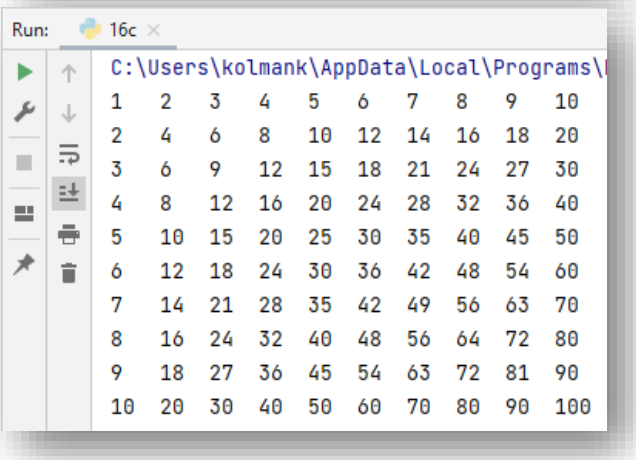


(16c.py)

Készítsünk programot, melyben egy kétdimenziós tömbben szorzótáblát készítünk, majd kiíratjuk a képernyőre!

- A program első sorában létrehozunk egy üres tömböt, szorzótábla néven.
- Majd (2-6. sor) feltöltjük tömbünket. Egymásbaágyazott külső for ciklussal indulunk. Az „i” változóval a sorokban lépkedünk, ezért minden sor elején kell egy „belső” tömböt létrehozni, melyben a „j” változós „belső” for ciklussal az „oszlopok” elemeit töltjük fel az ötödik sorban. Mivel a lista leme nullával indul, ezért hozzá kell adni egyet a változók aktuális értékéhez.
- Aztán minden feltöltött sor[] tömböt hozzáadunk a „szorzotabla” tömbhöz (6. sor).
- Végül tabulátorokkal elválasztva, egymásbaágyazott for ciklussal kiíratjuk a képernyőre a szorzótáblát.

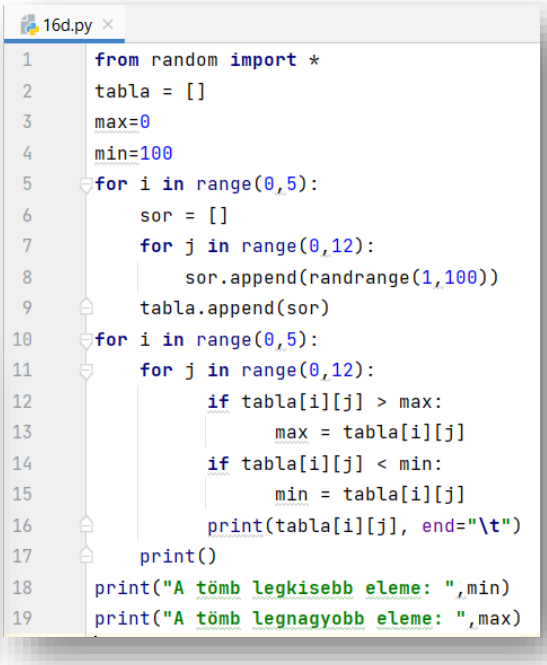


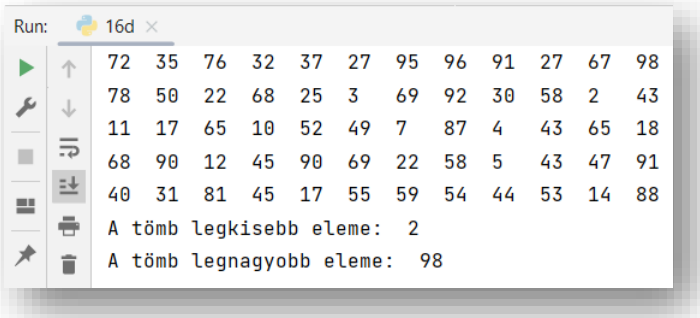


(16d.py)

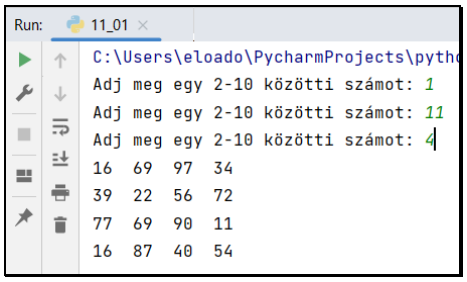
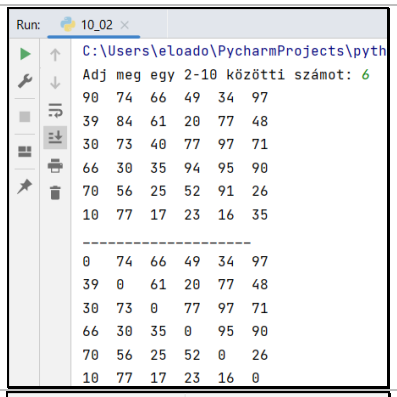
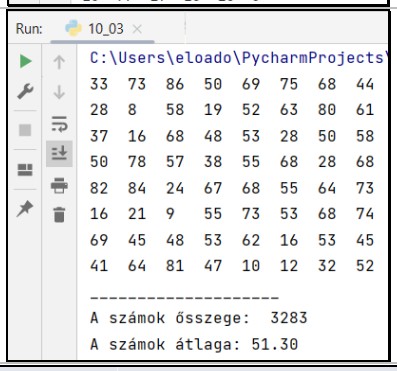
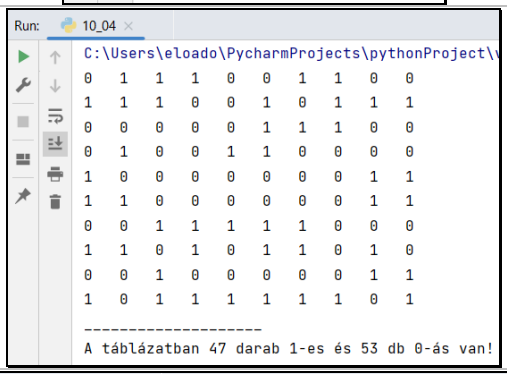
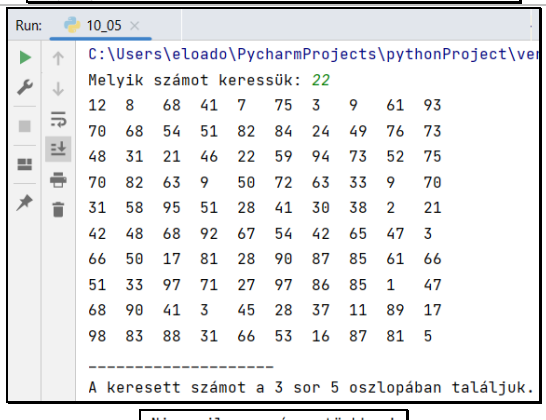
Készítsünk programot, melyben feltöltünk egy 5*12-es tömböt 1-99 közötti véletlen számokkal, majd kiíratjuk a tömb legkisebb és legnagyobb elmemét a minta alapján!

Tulajdonképpen az előző programot kell átírni és kibővíteni.





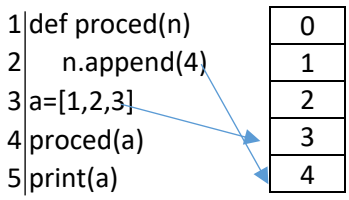
GYAKORLATI FELADATOK (16. témakörhöz)

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (11_01.py) Készíts egy programot, amely bekér egy „n” változót 2 és 10 között! Majd feltöltünk egy n*n-es tömböt 10 és 99 közötti véletlen számokkal! Végül kiíratjuk az n*n-es tömbböt képernyőre!	
2. (11_02.py) Mentsd el másként az előző programot, és bővítsed ki úgy, hogy az eredeti tábla kiíratása után, cseréld le az átlóban lévő számokat nulla értékre, majd újból írasd ki a tömböt!	
3. (11_03.py) Készíts programot, melyben egy 8*8 elemű tömbben generálsz 8-88 közötti véletlen számokat! Majd kiíratod a számok összegét és a számok átlagát, a minta szerint!	
4. (11_04.py) Készíts programot, melyben generálsz egy 10*10-es kétdimenziós tömböt, melyben generálsz 0-kat és 1-eseket véletlenszerűen! Majd megszámolod, hogy hány darab egyes és nulla található a tömbben és kiíratod a minta szerint.	
5. (11_05.py) Készíts python kódot, melyben a program az elején megkérdezi, hogy melyik számot szeretnéd megkeresni. A szám legyen 1 és 100 közötti. Majd generáljuk egy 10*10-es tömbbe 1 és 100 közötti véletlen számokat. Végül keressük meg hogy melyik sor melyik oszlopában találja meg a számot, vagy éppenséggel „Nincs ilyen szám a tömbben!”, akkor azt írja ki!	 Nincs ilyen szám a tömbben!

17.) LISTÁK AZ ELJÁRÁSOKBAN ÉS FÜGGVÉNYEKBEN

A függvényeknél megtanultuk, hogy egy értéket adnak vissza a főprogramnak. **A python nyelv lehetőséget ad arra, hogy az érték az összetett legyen, mondjuk egy lista.** Tehát át is tud venni egy listát és vissza is tud adni egy listát.
A cím (referencia) szerinti paraméteradásnál **a főprogram és az alprogram a változó nevéhez ugyanazt a memória területet rendeli hozzá.** Tehát **használhatunk két nevet ugyanannak a listának.**

Nézzünk egy példát:



- A jobb oldali szövegdobozban látszik, hogy a 3. sorban kezdődik a végrehajtás, mert az a főprogram első sora.
- Az „a” nevű változóba az [1,2,3] listát tesszük.
- A 4. sorban meghívjuk a proced nevű eljárást, az „a” változót használva paraméterül.
- A program végrehajtása az 1. sorban folytatódik, ahol az „a” változó memóriaterületéhez fog csatlakozni a most létrejövő „n” változó, ami a korábban létrehozott listát tarzalmazza.
- A 2. sorban hozzáfűzünk az „a” listához egy új elemet, a 4-et. Mivel csak egy listánk van két névvel, ezért az „a” listánk is módosul.
- A program végrehajtása, a főprogram 6. sorába ugrik, ahol kiíratjuk az „a” változót. Mivel az „a” változó ugyanaz mint az „n” változó, csak, ezért az eredmény a [1,2,3,4] lista.

Tehát ha az eljárás vagy függvény paramétere összetett változó, például lista, az átadás referencia szerinti, így a főprogramban található változót az alprogram módosíthatja. Ha a változó egyszerű változó, vagy konkrét érték, az átadás érték szerinti, tehát a főprogramban található változó nem módosul.
Ha egyszerre adunk át listát és egyszerű változót, akkor mindkét átadás referencia szerinti lesz. A karakterlánc egyszerű változónak számít.

(17a.py)

Készítsünk egy programot, melyben feltöltünk egy listát véletlenszerű „X” és „O” karakterekkel!

- A feladat megoldásához használjunk eljárást és függvényt!
- Az első sorban elérhetővé tesszük a véletlenszámok generálását.
- Majd egy megjelenítést lehetővé tevő eljárást készítünk „kiiratas” néven. (3-5. sor)
- A szokásos módon def utasítással kezdünk majd megadjuk az eljárás nevét és zárójelben az átvett lista nevét és a kettőspontot a végére.
- For utasítással végig megyünk a listán és kiíratjuk az elemeket, szóközökkel elválasztva. Majd egy sortöréssel zárjuk az eljárást.
- Aztán elkészítjük az adatokat elkészítő modult 8-16. sorban. Ebben az esetbe egy függvényt hozunk létre.
- A „generalas” nevű függvényben véletlenszám generálás segítségével a minta szerint feltöltjük a listánkat!
- A program fő részében (18-20. sor) már csak létrehozunk egy üres listát, amit a következő sorban feltöltünk a függvény meghívásával!
- Végül a kiíratő eljárás meghívásával kiíratjuk az feltöltött listát!

```
17a.py
1  from random import *
2
3  def kiiratas(ki_lista):
4      for i in ki_lista:
5          print(i,end=' ')
6      print()
7
8  def generalas(c):
9      gen_lista=[]
10     for i in range(c):
11         elem=randrange(0,2)
12         if elem % 2 ==0:
13             gen_lista.append("X")
14         else:
15             gen_lista.append("O")
16     return gen_lista
17
18 fo_lista=[]
19 fo_lista= generalas(10)
20 kiiratas(fo_lista)
```

```
Run:  17a
C:\Users\eloado\PycharmPro
O X O O O X O X X X
```

(17b.py)

Készítsünk programot, melyben generálunk egy 10 elemű listát, amiben 0 és 100 közötti véletlen számokat helyezünk el!
Kiíratjuk ezt a listát az első sorba!
Aztán rendezzük növekvő sorrendbe ezt a listát és azt is kiíratjuk!
Majd a lista első és utolsó elemét cseréljük meg, végül ezt is kiíratjuk a harmadik sorba!
A feladat megoldásánál alkalmazzunk eljárásokat és függvényeket!

```python
from random import *
def kiiratas(ki_lista):
    for i in ki_lista:
        print(i, end=", ")
    print()
def generalas(n):
    belso_lista= []
    for i in range(n):
        elem=randrange(0,100)
        belso_lista.append(elem)
    return belso_lista
def csere(belso_lista_2):
    n=len(belso_lista_2)-1
    belso_lista_2[0],belso_lista_2[n]=belso_lista_2[n],belso_lista_2[0]
kulso_lista=[]
kulso_lista=generalas(10)
kiiratas(kulso_lista)
kulso_lista.sort()
kiiratas(kulso_lista)
csere(kulso_lista)
kiiratas(kulso_lista)
```

```
Run: 17b
E:\00_MM\12_Python_programozas\programo
96, 69, 94, 73, 42, 9, 66, 27, 90, 12,
9, 12, 27, 42, 66, 69, 73, 90, 94, 96,
96, 12, 27, 42, 66, 69, 73, 90, 94, 9,

Process finished with exit code 0
```

Először értelmezzük a programot a képernyőkép alapján!
Majd gépeljük be és teszteljük!

(17c.py)

Készítsünk programot, melyben listákkal dolgozunk eljárásokban és függvényekben!

- töltsünk fel egy 2D tömböt szorzótáblával
- írassuk ki
- adjuk össze a szorzótábla értékeit
- duplázzuk meg a szorzótábla eleminek értékét

Értelmezzük, gépeljük be, futtassuk a programot!

```python
def tabla_feltotlt():
    for i in range(0,10):
        sor = []
        for j in range(0,10):
            sor.append((i+1)*(j+1))
        szorzotabla.append(sor)
def kiirat():
    for i in range(0,10):
        for j in range(0,10):
            print(szorzotabla[i][j], end="\t")
        print()
def osszead(szumma):
    for i in range(0,10):
        for j in range(0,10):
            szumma=szorzotabla[i][j]+szumma
    return szumma
def duplaz():
    for i in range(0,10):
        for j in range(0,10):
            szorzotabla[i][j]=szorzotabla[i][j]*2
osszeg=0
szorzotabla = []
tabla_feltotlt()
kiirat()
ki=osszead(osszeg)
print(ki)
print("_____")
duplaz()
kiirat()
ki=osszead(osszeg)
print(ki)
```

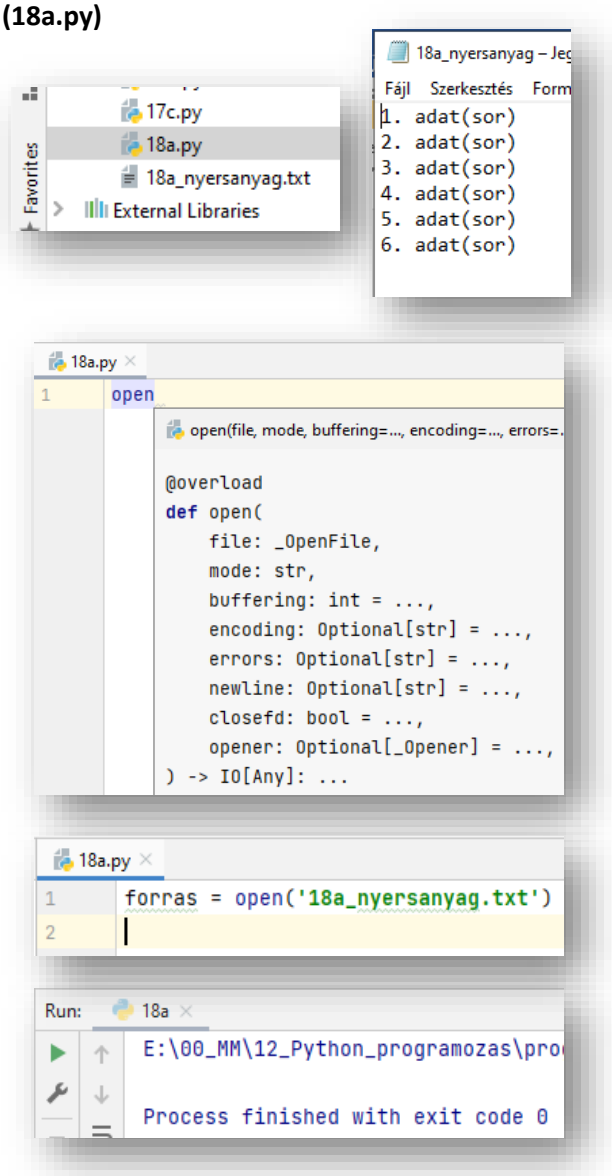
```
Run: 17c
E:\00_MM\12_Python_programozas\programok\v
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48  56  64  72  80
9   18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
3025
_____
2   4   6   8   10  12  14  16  18  20
4   8   12  16  20  24  28  32  36  40
6   12  18  24  30  36  42  48  54  60
8   16  24  32  40  48  56  64  72  80
10  20  30  40  50  60  70  80  90  100
12  24  36  48  60  72  84  96  108 120
14  28  42  56  70  84  98  112 126 140
16  32  48  64  80  96  112 128 144 160
18  36  54  72  90  108 126 144 162 180
20  40  60  80  100 120 140 160 180 200
6050
```

18.) FÁJLKEZELÉS PYTHONBAN

Az adatok tárolásának nagy jelentősége van a programozásban.
Eddig az adataink csak addig éltek, amíg a programunk futott. Amikor a program futását befejeztük, az addig használt adatok elvesztek.
Ha tartósan szeretnénk rögzíteni adatainkat, annak két módszere van. Az egyik, ha fájlban tároljuk ezeket, a másik, ha adatbázist használunk. Ebben a fejezetben az elsőt alkalmazzuk.
Nézzük meg, hogy egy python programból hogyan érhetünk el egy **txt** vagy **csv** fájlt, tehát hogyan tudunk beolvasni, majd hogyan tudjuk feladatunk végeztével kiíratni txt, csv fájlba a végeredményt.
(A következő feladatok megoldásához előkészített nyersanyagokat (txt, csv) használunk fel.)

Adatok beolvasása

(18a.py)

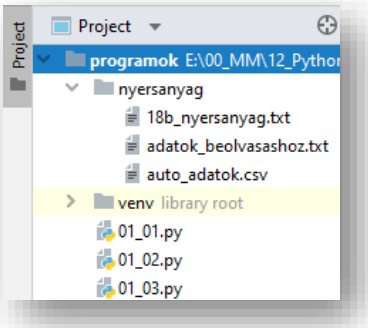


Készítsünk programot, melyben **beolvasunk** a programba egy előre elkészített txt fájlt!
Ennél a feladatnál a **forrásfájlt és a programfájlt azonos mappában helyezzük el**!

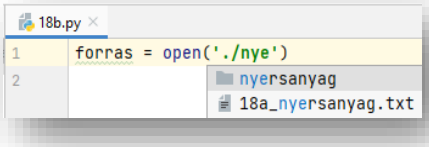
- Először másoljuk a nyersanyagot abba a mappába, ahol a programfájljainkat tároljuk.
- Majd hozzuk létre mellé a 18a.py program fájlt!
- A fájl elérése és megnyitása python programozásban az **open függvény** segítségével történik.
- Ha PyCharb-ban dolgozunk, akkor Ctrl + Shift +I billentyűkombinációval elérhetjük az **esetleges paraméterek**et, amelyeket felhasználhatunk.
- Mi a file, a mode, és az encoding –ot fogjuk használni elsősorban.
- Tehát az **open parancs megadása után zárójelek között először sztring formátumban megadjuk a fájl pontos nevét**. Mivel azonos mappában van a nyersanyag és a forrásfájl, ezért csak aposztrófok között megadjuk a fájl nevét, kiterjesztéssel együtt!
- Az előző parancs létrehoz egy objektumot. Ahhoz hogy dolgozni tudjuk **egy változóba kell tennünk ezt az objektumot**. Ezért elé írunk egy változó nevet, jelen esetben „forras” néven.
- Az, hogy sikeres volt a megnyitás, az úgy fog kiderülni számunkra, ha lefuttatjuk a programot, és nem kapunk hibajelzést.
- Tehát ebben az esetben semmi látványos dolog nem történik, csak egyszerűen megnyitottunk egy txt fájlt!

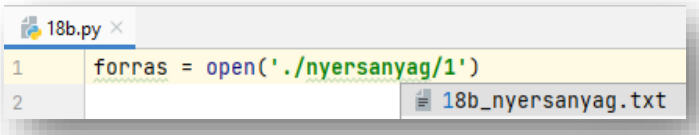
Az előző példában a forrásfájl azonos mappában volt a programfájllal. De programozás során általában több nyersanyaggal, fájlal dolgozunk. Ezért célszerű külön almappában tárolni ezeket a fájlokat. Nézzünk erre példákat!

(18b.py)



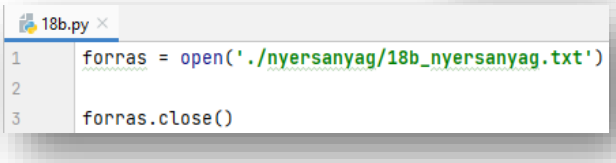
- Hozzunk létre egy „nyersanyagok” nevű mappát az eddigi programfájljaink mellé!
- Tegyük bele a 18b_nyersanyag.txt nevű fájlt a nyersanyag mappába!
- Ebben az esetben már nem elég csak a fájl nevét megadni, hanem ki kell egészíteni a **fájl elérési útjá**val!
- A könyvtárstruktúra arra a szintjére kell hivatkozni, ahol a fájl most található.
- A PyCharm segít a nyersanyag nevének kiválasztásában. **A hely megadásának az elején „./” után adjuk meg az almappa nevét, majd „/” karakter után a fájl nevét!**





Találkozhatunk olyan esettel, hogy felfelé kell haladnunk a mappánkban. Ez akkor fordulhat elő, ha programfájlunk helyezkedik el az almappában és a nyersanyag a főmappában van. Ilyen esetben „./../valami.txt” módon érhetjük el a nyersanyagunkat. Tehát két pont karakterrel haladunk egy mappányit felfelé!

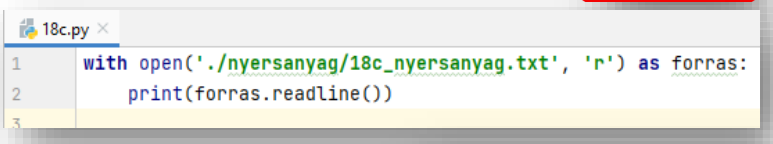
Nagyon fontos, ha megnyitunk egy nyersanyag fájlt, azt be is kell zárni.
Ennek akkor van jelentősége, ha nem csak olvassuk a fájlt, hanem írjuk is.

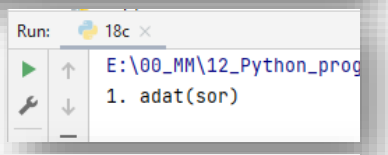


(18c.py)

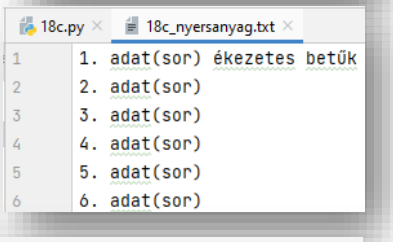
A pythonban létezik egy úgynevezett **context manager**, amellyel **egyszerűbben dolgozhatunk, ezért mi a későbbiekben ezzel fogunk dolgozni**!

- Ebben az esetben a „**with**” kulcsszóval kezdünk.
- Majd jöhet az **open függvény** az elérési úttal, mely után beírjuk az **„as” utasítást és az objektum nevét**.

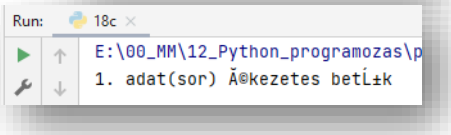




- Ebben az esetben nem kell bezárnunk a megnyitott fájlt, akár olvasásról, akár írásról van szó.
- Ha **csak olvasásra (read)** szeretnénk megnyitni a nyersanyagunkat, akkor az open utasításon belül, a fájl helyének megadása után vesszővel elválasztva megadjuk az **’r’ paramétert**.
- Aztán **olvassuk ki az első sor tartalmát a fájlból a readline() utasítással!**



- Ha a 18c_nyersanyag.txt fájl első sorába beírunk olyan szöveget, amely ékezetet tartalmaz, majd bezárás után futtatjuk a programunkat, akkor látszik, hogy hibásan értelmezi az ékezetes betűket.



- Az előző probléma megoldására egy **encoding='utf-8'** paraméterrel egészítjük ki az open függvényünket! (Ez Windows operációs rendszernél okoz problémát.)
- Az előző readline() utasítást másoljuk még le háromszor és futtasuk újból a programot!
- Itt látszik a, hogy a txt-ben lévő sor végi (most felesleges „/n') enterket is beolvassa. Üres sorokat is kapunk. (A későbbiekben, majd nézünk erre megoldást.)
- Van egy úgynevezett **fájlmutató,** ami „megmondja" nekünk bájtban, hogy éppen hol állunk a program futása során. Ehhez a **tell()** metódust használjuk. Ezeket az utasításokat másoljuk be a mintán látható helyekre, és futtassuk a programot!
- Azt látjuk, hogy kiírja a képernyőre, hogy hányadik karakternél járunk. Az eredmények bitszámokat jelentenek.
- Nem csak kiírathatjuk a helyeket, hanem **„ugrálhatunk" a pozíciók között a seek() utasítással**! Írjuk be a negyedik sorba az utasítást, és adjuk meg, hogy ugorjon vissza a 0. karakterpozícióba!
- Így ha futtatjuk a programot láthatjuk, hogy kétszer lesz kiírva az első sor!

18c.py

```
with open('./nyersanyag/18c_nyersanyag.txt', 'r', encoding='utf-8') as forras:
    print(forras.readline())
```

Run: 18c

```
E:\00_MM\12_Python_programozas\
1. adat(sor) ékezetes betűk
```

18c.py

```
with open('./nyersanyag/18c_nyer
    print(forras.tell())
    print(forras.readline())
    forras.seek(0)
    print(forras.tell())
    print(forras.readline())
    print(forras.tell())
    print(forras.readline())
    print(forras.readline())
```

Run: 18c

```
E:\00_MM\12_Python_programoz
0
1. adat(sor) ékezetes betűk

0
1. adat(sor) ékezetes betűk

31
2. adat(sor)
```

(18d.py)

A következő programban nyissuk meg a 18d_nyersanyag.txt fájlt a tanult módon.
A **readlines()** utasítással az összes sor beolvasásra kerül és egy listában megjelenítődik. Egy-egy sor, egy-egy lista elem, sztring formátumban. (A sor végi „/n" vezérlő karakterek is látszódnak.)

19d.py

```
with open('./nyersanyag/18d_nyersanyag.txt', 'r', encoding='utf-8') as forras:
    print(forras.readlines())
```

Run: 19d

```
E:\00_MM\12_Python_programozas\programok\venv\Scripts\python.exe E:/00_MM/12_Python_programozas/pro
['Kiss Balázs\n', 'Nagy Imre\n', 'Horváth Béla\n', 'Kovács Éva\n', 'Soós Mária\n', 'Balogh Ágota']
```

(18e.py)

Mentsük le másként az előző programot, és csak írjuk át a readlines() utasítást **read()**-re! Ebben az esetben egymás alá íratjuk ki a fájl sorait!
Ezen **utasításoknak a hátránya, hogy memória túlcsorduláshoz is vezethet**. **Ezért** nem ezeket az utasításokat fogjuk használni, hanem a **for ciklus segítségével soronként dolgozunk**!

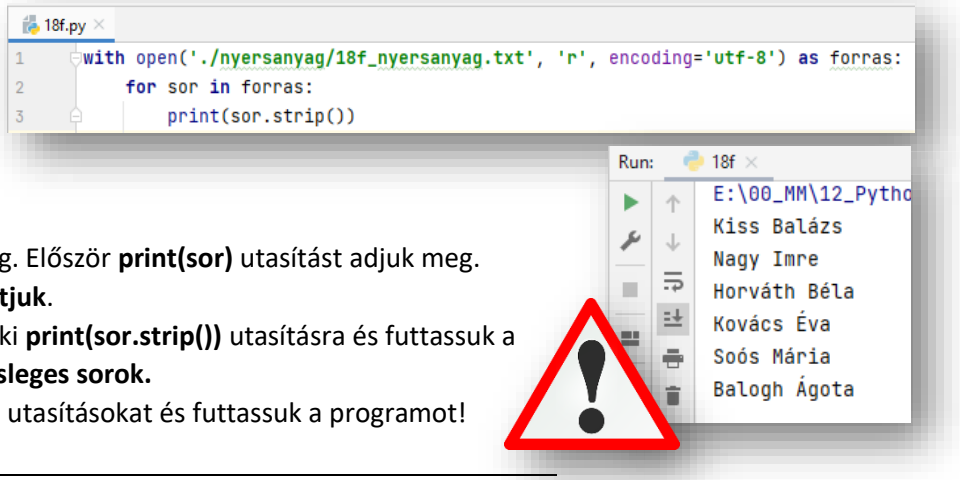
19e.py

```
with open('./nyersanyag/18d_nyersanyag.txt', 'r', encoding='utf-8') as forras:
    print(forras.read())
```

Run: 19e

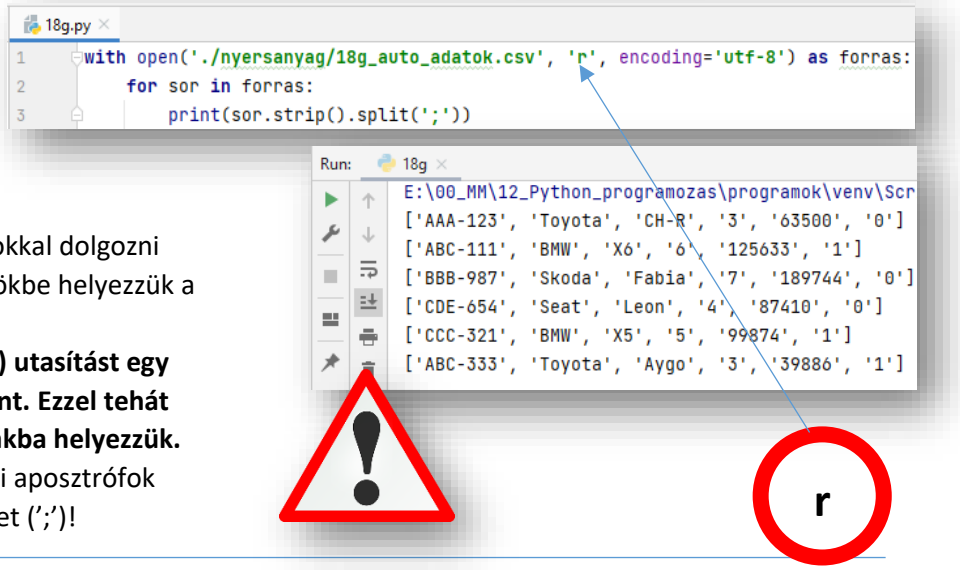
```
E:\00_MM\12_Pythor
Kiss Balázs
Nagy Imre
Horváth Béla
Kovács Éva
Soós Mária
Balogh Ágota
```

(18f.py)



Tehát a memória túlcsordulás kivédésére soronként olvassuk be a 18f_nyersanyag.txt szövegfájl karaktereit.
Ezt a for utasítással tesszük meg. Először **print(sor)** utasítást adjuk meg.
Ekkor még az **üres sorokat is látjuk**.
Ennek „kivédésére” egészítsük ki **print(sor.strip())** utasításra és futtassuk a programunkat. **Eltűnnek a felesleges sorok.**
A minta alapján értelmezzük az utasításokat és futtassuk a programot!

(18g.py)

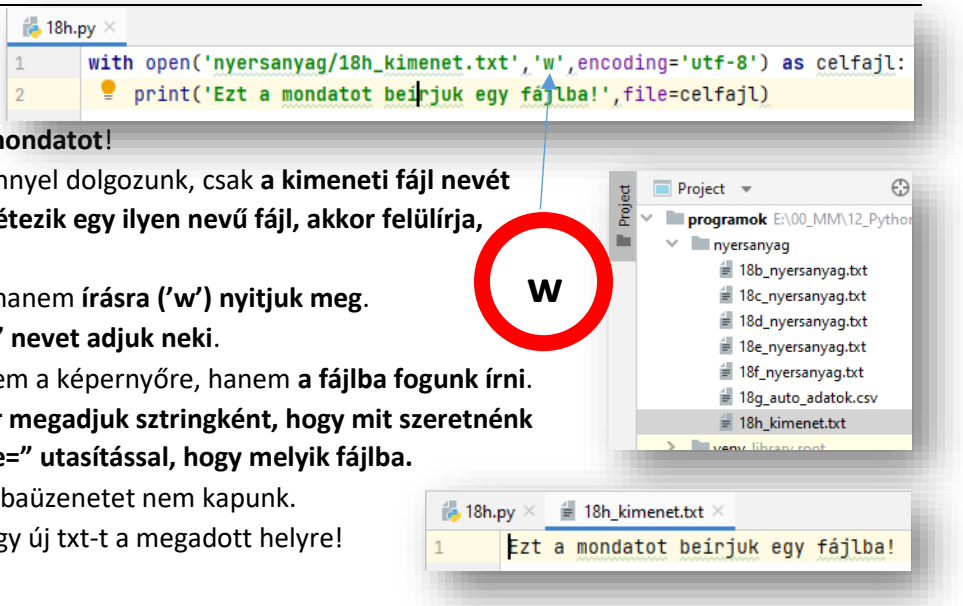


A txt fájlokon kívül **csv fájlokkal is tudunk dolgozni**. Olvassuk be a 18g_auto_adatok.csv fájlt soronként.
Ahhoz, hogy a beolvasott adatokkal dolgozni tudjunk a későbbiekben, tömbökbe helyezzük a sztringeket.
Egészítsük ki a print(sor.strip()) utasítást egy split() utasítással a minta szerint. Ezzel tehát feldaraboljuk sorainkat és listákba helyezzük.
A zárójelek között meg kell adni aposztrófok között az elválasztó karaktereket (’;’)!

Adatok kiírása fájlba, adatok módosítása

Az előzőekben megismerkedhettünk azzal, hogy hogyan tudunk fájlból adatokat beolvasni. Most pedig megnézzük, hogy hogyan tudunk a programunkból fájlba írni, menteni és így ezek az adatok képesek megmaradni a programokból való kilépés után is.

(18h.py)

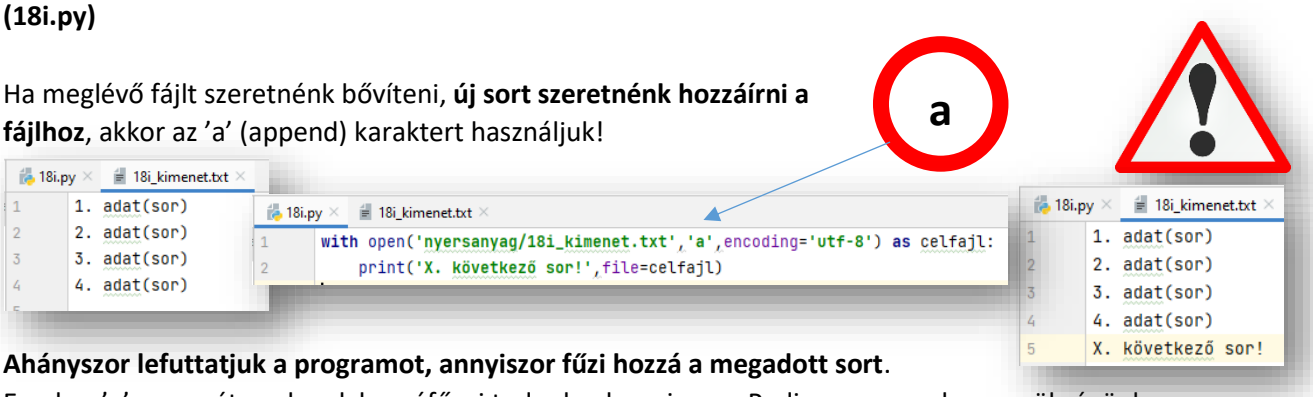


A következő programban egy **txt fájlba fogunk kiíratni egy mondatot**!

- Ugyanúgy az open függvénnyel dolgozunk, csak **a kimeneti fájl nevét és helyét adjuk meg. Ha létezik egy ilyen nevű fájl, akkor felülírja, ha nem akkor létrehozza!**
- Most nem olvasásra (’r’), hanem **írásra (’w’) nyitjuk meg**.
- Ebben az esetben **„celfajl” nevet adjuk neki**.
- A print utasítással most nem a képernyőre, hanem **a fájlba fogunk írni**.
- **A zárójelek között először megadjuk sztringként, hogy mit szeretnénk a fájlba íratni, majd a „file=” utasítással, hogy melyik fájlba.**
- Futtassuk a programot! Hibaüzenetet nem kapunk.
- Látjuk, hogy létrehozott egy új txt-t a megadott helyre!

(18i.py)

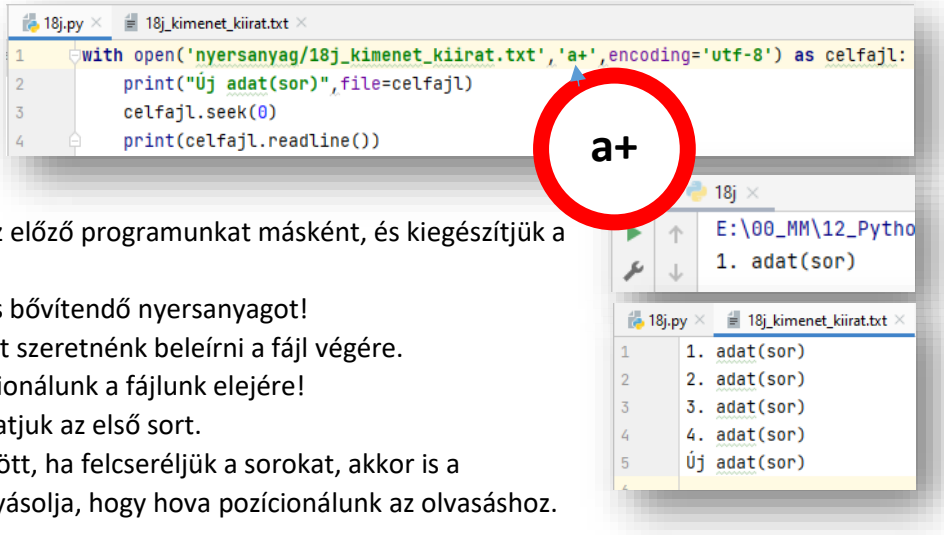
Ha meglévő fájlt szeretnénk bővíteni, **új sort szeretnénk hozzáírni a fájlhoz**, akkor az ’a’ (append) karaktert használjuk!



Ahányszor lefuttatjuk a programot, annyiszor fűzi hozzá a megadott sort.
Ezzel az ’a’ paraméterrel csak hozzáfűzni tudunk, olvasni nem. Pedig nagyon sokszor szükségünk van egyszerre a kettőre. Ezért a következő programban erre nézünk példát.

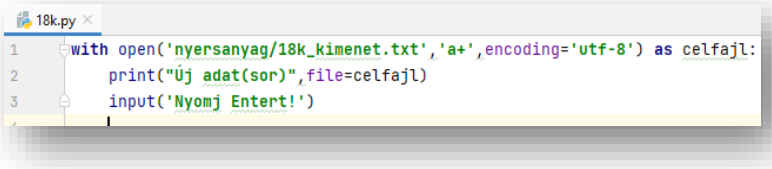
(18j.py)

Ha szeretnénk **fájlba íratni és képernyőre is íratni**,
akkor az ’a+’ paramétert kell alkalmaznunk.
Ebben az esetben elmentjük az előző programunkat másként, és kiegészítjük a mintán látható utasításokkal.
Tehát átírjuk a megnyitandó és bővítendő nyersanyagot!
Megadjuk a 2. sorban, hogy mit szeretnénk beleírni a fájl végére.
A harmadik sorban visszapozícionálunk a fájlunk elejére!
Majd readline() utasítással kiíratjuk az első sort.
A fájlmutató az olvasáshoz kötött, ha felcseréljük a sorokat, akkor is a fájl végére ír. Tehát nem befolyásolja, hogy hova pozícionálunk az olvasáshoz.

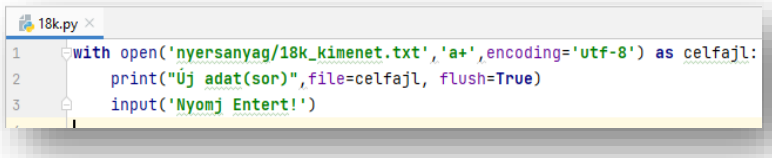


(18k.py)

Nézzük meg, hogy a **fájlba való kiíratás mikor történik** meg! A lefutás során, amikor a kiíratás sorához érünk, vagy a program lefutásának végén?

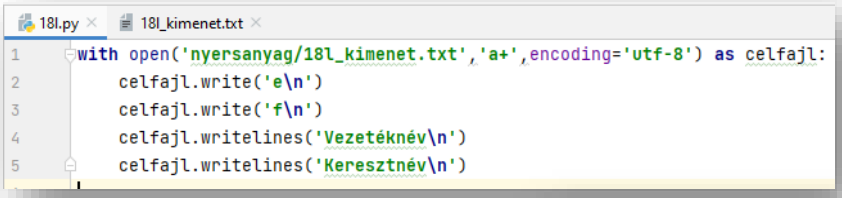


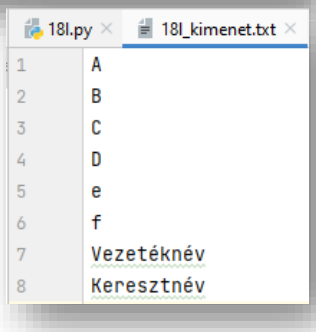
- Most fontos a műveletek sorrendje, úgyhogy haladjunk lépésről lépésre!
- Először gépeljük be a jobb oldalon lévő kódot!
- Az input() utasítással egy karakter lenyomására vár!
- Nyissuk meg a feladathoz tartozó nyersanyagot és a feladat lefutása során folyamatosan vizsgáljuk!
- Futtassuk a programot!
- Vár az enter lenyomására, a txt-ben még nem történt semmi.
- Amikor megnyomjuk az entert akkor kerül be az új sor a kimeneti fájl végére!
- Tehát kiderült, hogy nem azonnal menti a változásokat, hanem egy pufferben tárolja mindaddig, amig nem végzünk a program futásával.
- Ha azt szeretnénk, hogy azonnal legyen mentve, akkor egy újabb paramétert kell megadnunk a második sorunkban. Ez pedig a flush=’True’ paraméter.
- Ha így futtatjuk a programunkat, akkor amikor a 2. sor lefut azonnal bekerül a txt-be az új sor.



(18l.py)

Egy **másik példa a fájlba való íratásra**!
Használjuk write() és writelines() utasításokat a minta szerint!

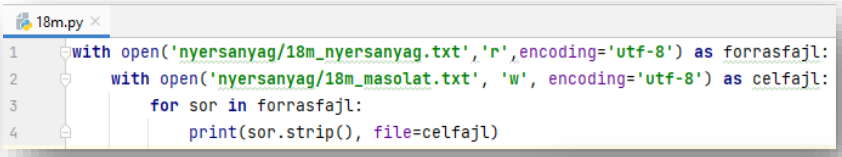


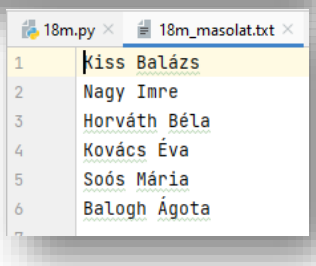


- Nyissuk meg a 18l_kimenet.txt fájlt írásra és olvasásra!
- Először próbáljuk ki a kódunkat „\n”-ek nélkül! Mi történik?
- Egymás mellé kiírja a karaktereket és a szavakat.
- Viszont ha használjuk a „\n” sortörést, akkor a képernyőképpel megegyező eredményt kapunk.

(18m.py)

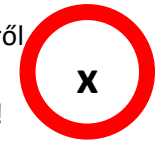
Nézzük meg, hogy **hogyan tudunk másolatot készíteni egy txt fájlról**!





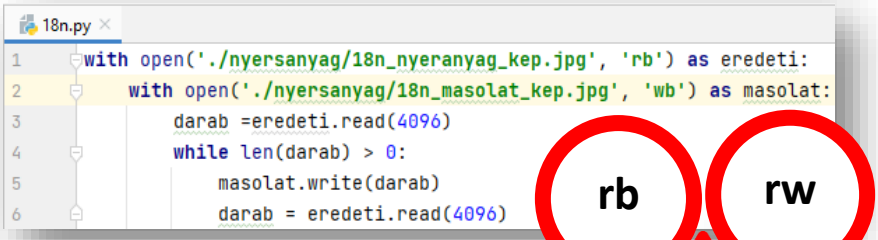
- A program első sorában megnyitunk olvasásra egy forrásfájlt!
- A második sorban megnyitunk írásra (új névvel) egy célfájlt!
- Soronként végig megyünk for utasítással a fájlunkon.
- A negyedik sorban a célfájlba tesszük – plusz enterek nélkül – az adatokat.
- Ellenőrizzük, hogy elkészült-e a másolat!

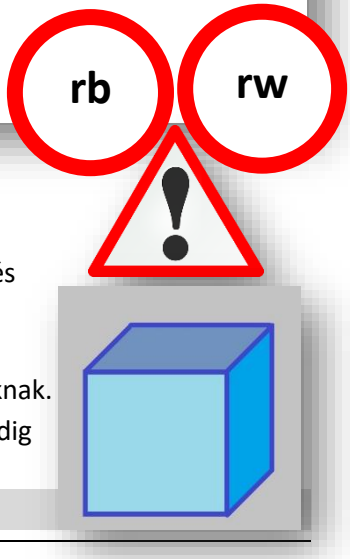
Eddig beszéltünk az r(olvasás) / w(írás) / a(végére írás) / a+(végére írás + olvasás) paraméterekről.
Ha az ’x’ paramétert használjuk, akkor ha már létezik a fájl, akkor nem engedi felülírni!
Ha nem létezik, akkor létrehozza, ha létezik, akkor hibaüzenetet küld! Kivételkezelés szükséges!



(18n.py)

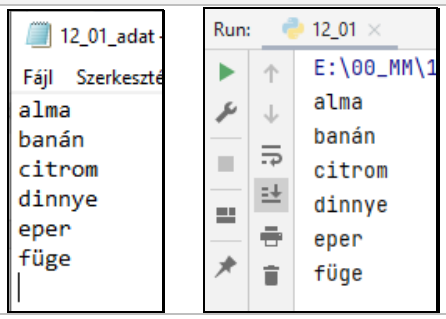
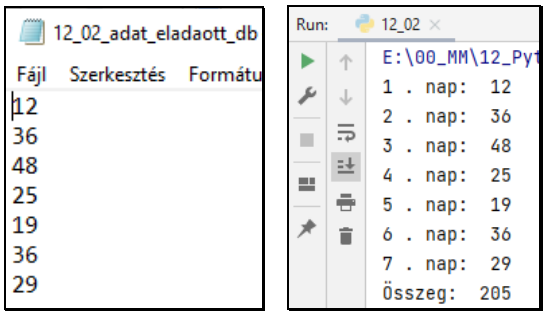
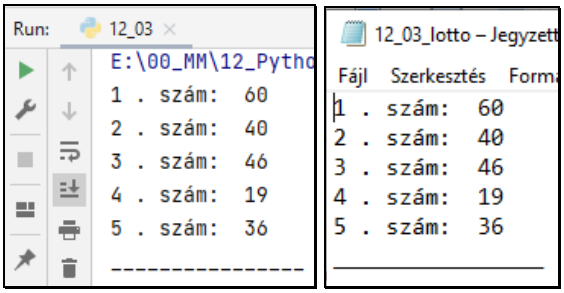
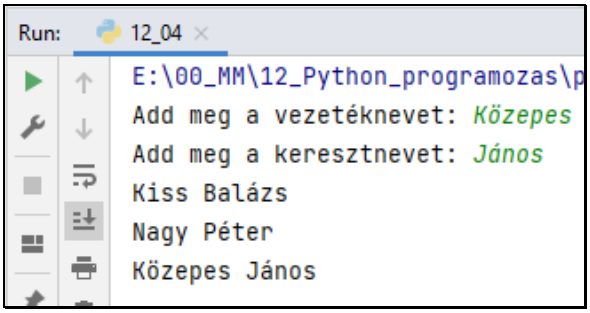
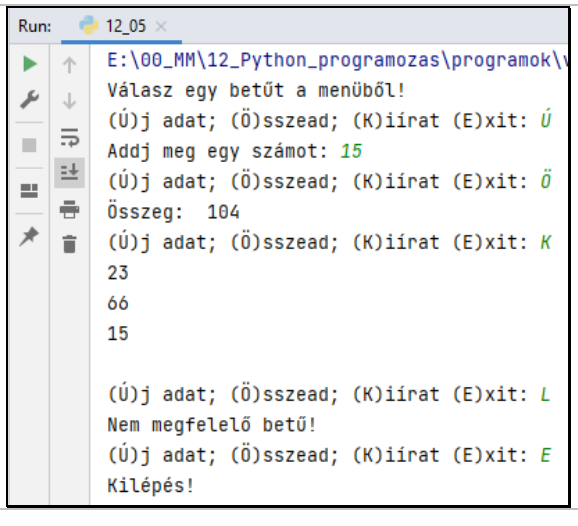
Másoljunk le egy bináris fájlt! Mondjuk egy jpg képet!





- Ebben az esetben megnyitjuk az eredeti képet olvasásra – az első sorban - , de ebben az esetben, a bináris megnyitás miatt **’rb’ paramétert adunk meg!**
- Aztán a másodiksorban megadjuk, hogy a másolatot milyen néven mentsük, és itt ’wb’ paramétert adunk meg!
- A túlcsordulás miatt darabonként dolgozzuk fel (másoljuk) az adatokat, ezért a harmadik sorban, egy darab változóba beolvassuk egy kisebb részét a fájlinknak.
- Aztán mindaddig amig be tudunk olvasni a fájlból biteke (amíg nem nulla), addig másolatot készítünk a másolat nevű fájlba.
- Futtassuk a programunkat, ellenőrizzük a másolatot!

GYAKORLATI FELADATOK (17-18. témakörhöz)

FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (12_01.py) Készítsd el először a mintán látható txt fájlt és mentsd el 12_01_adat.txt néven a nyersanyag mappába! Majd készíts programot, mellyel beolvasod és kiíratod a képernyőre a minta szerint, külön sorban!	
2. (12_02.py) Készítsd el először a mintán látható txt fájlt és mentsd el 12_02_adat_eladott_db.txt néven a nyersanyag mappába! Majd készíts programot, mellyel beolvasod és kiíratod a képernyőre a minta szerint, külön sorban, de közben egy „osszeg” nevű változóban összeadod a beolvasott számokat! Végül kiíratod a képernyőre az eredményt a minta szerint!	
3. (12_03.py) Készíts programot, melyben lottószámokat generálsz egy 12_03_lotto.txt nevű fájlba! A fájlba íratás a minta szerint történjen! Ha nincs ilyen fájl, akkor hozzon létre egyet! Közben a képernyőre is írja ki a számokat a minta szerint! Ha többször futtatom a programot, akkor a txt-be kerüljenek be az új számok is!	
4. (12_04.py) Készíts programot, melyben létrehozol egy 12_04_nevsor.txt nevű fájlt! Ebbe a fájlba vezetékneveket és keresztneveket kérjen be a program! Mindig csak egy személy nevét! Többször futtasd a programot, vigyél be több nevet! Majd mindig írasd ki a bővített listát a képernyőre a minta szerint!	
5. (12_05.py) Készíts programot, melyben egy menüből kell kiválasztani a minta alapján, hogy mi történjen! Egy 12_05_adat.txt nevű fájlban tároljon a program számokat! A begépelt betűk alapján tudjon a program új számot bevinni, a tárolt számok összegét kiíratni, illetve az összes eddig tárolt számot kiíratni a képernyőre a fájlból! A program mindaddig fusson, amig a felhasználó be nem gépeli az „E” betűt, amikor is az exit() parancsot hajtsa végre! Ha nem a négy betű közül írsz be egyet, akkor írja ki, hogy „Nem megfelelő betű!” A feladatokat eljárásokkal készítsed el!	

19.) SZÓTÁR ADATTIPUS

A **szótár adattípus** egy a programozásban nagyon gyakran használt típus. Nézzünk egy példát, melyben autók adatait tároljuk (rendszám, márka, típus, gyártási év, megtett kilométer, sérülésmentes-e).

Eddig ezt listában tettük meg, hiszen listában különböző típusú adatokat is tárolhatunk:

auto=['AA-BB-123','Audi','A4','2022','35000','True']

Ha a gyártási évére vagyunk kíváncsiak, akkor az **indexre hivatkozva** tudjuk megadni.

print(auto[3])

Ezzel van egy kis nehézség mert, ha nagyon sok adatot tárolunk, és azzal kell dolgoznunk, bonyolulttá válik a munka. Vagy ha valaki más olvassa a kódunkat, akkor nehezen igazodik ki a sok index között.

Ezért, sokkal **praktikusabb** lenne, ha **a különböző adatokra nem indexekkel hivatkoznánk, hanem valamiféle címkékkel, vagy megnevezésekkel azonosítanánk**. Erre van lehetőségünk a szótár használatánál.

```
auto={
        'rendszam' : 'AA-BB-123',       # a sztring típusokat aposztrófok közé tesszük
        'marka' : 'Audi',
        'tipus' : 'A4',
        'gyart_ev' : 2022,              # a szám típusnál nem kell aposztróf
        'km_ora' : 35000,
        'ser_e' : True,                 # a logikai típusnál sem kell aposztróf
}
```

A szótár nevének megadása után nyitó „kapcsos” zárójelet ({)kezdünk és a minta alapján aposztrófok között elkezdjük megadni a címkék nevét – ezt nevezzük **kulcs**nak(key) -, majd kettőspont után **érték**et (value) adunk. A sorok végén ne felejtsünk el vesszöket rakni! A szótár megadásának végén zárjuk a kapcsos zárójelünket (})

Adatokra hivatkozni úgy tudunk, hogy a címkék nevét adjuk meg:

print(auto['gyart_ev'])

Így egyszerűbb kódot írni, illetve kódot olvasni.

Mikor érdemes a szótár adattípust alkalmazni?

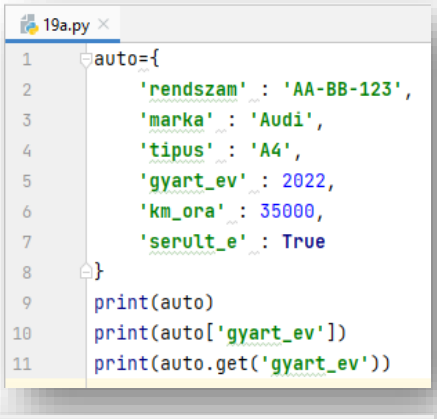
Ha homogén adatokat kell tárolni, mondjuk műszerrel mért adatokat, melyeket különböző időpontokban tettük meg, akkor elég a két dimenziós lista. (Például: hőmérséklet adatok reggel, délben, este)

De ha több, különböző típusú adatokat szeretnénk tárolni, akkor érdemes szótár típust használni.

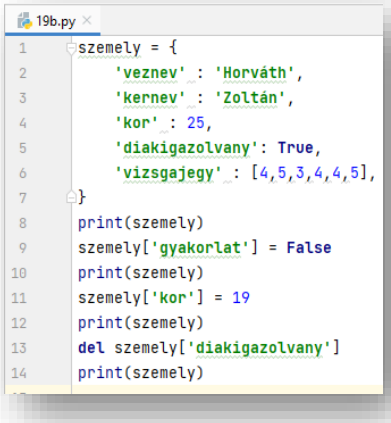
(19a.py)

Készítsünk programot, melyben az előzőekben látható példát valósítjuk meg.

- Egyszerűen létrehozunk egy „auto” nevű szótár típust a megadott adatokkal. (1-8. sor)
- Aztán kiíratjuk a képernyőre a teljes tartalmat kulcsokkal és értékekkel. (9. sor)
- A minta szerint a gyártási év értékének kiíratására két módszert is megnézünk.

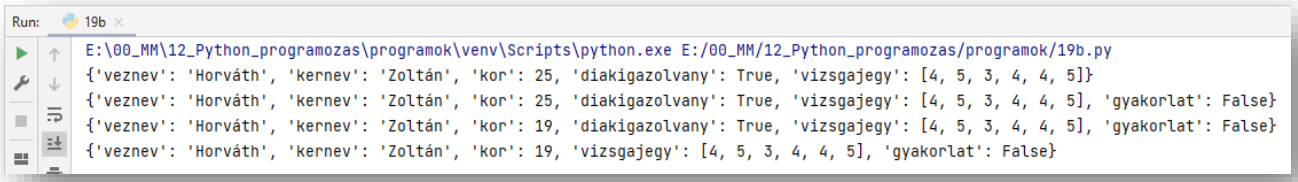


(19b.py)



Ebben a példában hozzunk létre egy „szemely” nevű szótárt, melyben egy főiskolai hallgató adatait tároljuk. (1-7. sor)

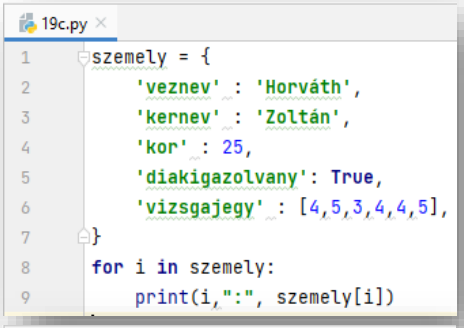
- Nézzünk pár hasznos utasítást, majd mindig írassuk ki a képernyőre, hogy lássuk a változást!
- Adjunk hozzá egy újabb mezőt „gyakorlat” néven, melynek logikai típusa legyen!
- Aztán változtassuk meg az egyik tárolt értéket! A hallgató korát írjuk át 19-re!
- Végül töröljünk egy felesleges mezőt a del paranccsal a minta szerint!

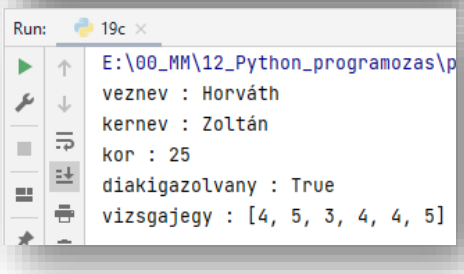


(19c.py)

Szótár bejárása (1.):

- Hozzunk létre egy új programot, melybe másoljuk át a „szemely” szótár deklarálását! (1-7. sor)
- Ebben a rövid programban a bejárás egyik módját nézzük meg for ciklussal. (8. sor)
- Tehát kiíratjuk a kulcsot és a hozzá tartozó értéket. A kettő közé beszúrunk egy kettőspontot! (9. sor)

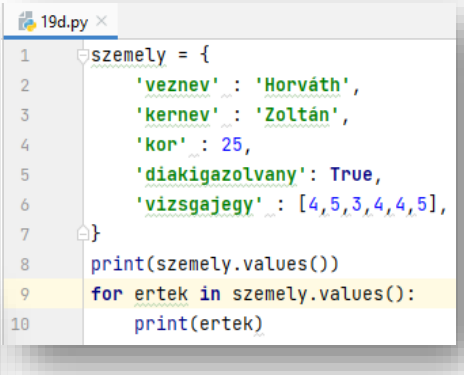


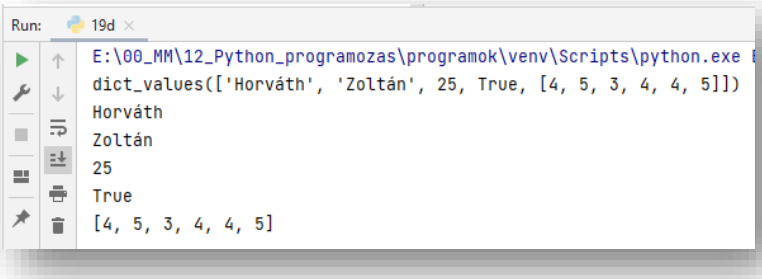


(19d.py)

Szótár bejárása (2.):

A szótár bejárásának másik módja a **values()** használata.
Tehát ha a szótárra közvetlenül meghívjuk ezt a metódust (8. sor), akkor az adatokat egy listában tárolva jeleníti meg.
A 9-10. sorban for ciklussal soronként meghívjuk az értékeket és kiíratjuk a képernyőre.





A legtöbb esetben nem csak egy ember adataival kell dolgoznunk, hanem – maradva az előző példánál – egyszerre több egyetemi hallgató adataival kell feladatokat elvégeznünk. Tehát képzeljük el, hogy egy listában tároljuk a különböző személyek adatait.

hallgatok=[{személy_0},{személy_1},{személy_2},{személy_3},…,{személy_n}]

Tehát ebben a listában ha egy diák adataira akarunk hivatkozni, akkor indexére hivatkozunk.

print(hallgatok[2])

Ha egy adott diák konkrét adatára, akkor az index után megadjuk a kulcsot is, amelyre kíváncsiak vagyunk.

print(hallgatok[2]['kor'])

(19e.py)

A szótár adattípus használatára nézzünk egy komolyabb, bonyolultabb példát!
Készítsünk programot, melyben egy txt fájlból beolvasunk adatokat, amelyeket szótár adattípusba helyezzük el. Aztán feladatokat végzünk a szótárakba helyezett adatokkal.
A példában egyetemi hallgatók adataival fogunk dolgozni.

- Először **hozzunk létre egy üres „adatok" nevű listát**, melyben majd tárolni fogjuk soronként a kiolvasott adatokat. (1. sor)
- Aztán **megnyitjuk a nyersanyag txt –t olvasásra** (2. sor), majd soronként hozzáadjuk az adatok listához (3-4. sor). Az adatok listába az **append() felhasználásával hozzáadjuk a beolvasott sorokat, felesleges soremelés karakterek nélkül (strip()).**
- A program írása közben mindig ellenőrizzük print() utasítással, hogy sikerült-e az előző művelet. Tehát létrejött-e az „adatok" feltöltött lista? Aztán a későbbiekben kettőskereszt (#) elé írásával, kommentbe helyezhetjük.
- A következő sorokban **elkészítjük a szótár típust**. (6-19. sor) Fel **kell építenünk az adatszerkezetet, és fel is kell töltenünk adatokkal.**
- A 6. sorban **létrehozzuk a „diak" nevű üres szótárt**. Ebben tároljuk majd az egyes diákok adatait.
- A 7. sorban pedig **egy „diakok" üres listát hozzuk létre**, melybe majd az összes diák szótár adatait rakjuk bele.
- For ciklussal be kell járnunk az adatok nevű listánkat, hiszen ebbe olvastuk be a nyersanyagot a txt-ből.
- Ha megnézzük a listánk elemeit, azok gyakorlatilag sztringek, szóközökkel elválasztva.

```
['Kiss János 20 történelem 0', 'Nagy Béla 19 matematika 1', 'Horváth Éva 21 biológia 1', 'Kovács
```

- Tehát egy-egy ilyen **sztringet fel kell darabolnunk kisebb részekre, a space-ek mentén**, ahol meg tudjuk adni a típusokat is.
- A feldaraboláshoz **létrehozunk egy listát „diak_adatok" néven (9. sor), melyben a split() metódus segítségével feldaraboljuk**. (Ha nem adunk meg argomentumot a zárójelek között, akkor az alapértelmezett space-ek mentén darabolja fel a sztringünket.)
- print(diak_adatok) paranccsal ideiglenesen tesztelhetjük is, hogy jól dolgoztunk-e. Aztán ezt a sor töröljük.
- Aztán kezdődhet a szótár illetve a lista felöltése. A minta szerint **megadjuk a kulcsokat és a hozzá tartozó „diak_adatok" megfelelő indexét**.
- A „kor" kulcsnál az étréket **int()** paranccsal számmá alakítjuk.
- A kollégiumhoz tartozó 1 illetve 0 számoknál „**if" feltétellel** True vagy False értéket állítunk be.
- A 18. sorban a **diakok.append(diak)** paranccsal becsatoljuk a „diakok" listához az aktuális sort.
- Végül **ki kell ürítenünk a „diak" szótár aktuális adatait** a 19.sorban.
- Aztán nézzük meg, hogy mit tartalmaz a „diakok" nevű lista. (20. sor) A listán belül vannak a szótárak sorban egymás után. És az egy-egy szótárban a gyerek adatait egymás után látjuk.

```
[{'vezeteknev': 'Kiss', 'keresztnev': 'János', 'kor': 20, 'szak': 'történelem', 'kollegista': False}, {'vezet
```

- Tehát sikerült létrehozni az adatszerkezetet és sikerült feltölteni azt.

19e.py

```
1  adatok = []
2  with open('./nyersanyag/19e_szemelyi_adatok.txt','r',encoding='utf-8') as fajl:
3      for sor in fajl:
4          adatok.append(sor.strip())
5  #print(adatok)
6  diak={}
7  diakok=[]
8  for elem in adatok:
9      diak_adatok = elem.split()
10     diak['vezeteknev'] = diak_adatok[0]
11     diak['keresztnev'] = diak_adatok[1]
12     diak['kor'] = int(diak_adatok[2])
13     diak['szak'] = diak_adatok[3]
14     if diak_adatok[4]=='1':
15         diak['kollegista']=True
16     else:
17         diak['kollegista'] = False
18     diakok.append(diak)
19     diak={}
20 #print(diakok)
21 # a matematika szakos hallagtók neveinek kiíratása
22 print("\n----- Matematika szakos hallagatók listája: -----")
23 for diak in diakok:
24     if diak['szak']== 'matematika':
25         print(diak['vezeteknev'] + ' ' + diak['keresztnev'])
26 # tegyük listába történelem szakos hallgatók adatait
27 print("\n----- Történelem szakos hallagatók adatai: -----")
28 tortenelem_szak=[diak for diak in diakok if diak['szak'] == 'történelem']
29 print(tortenelem_szak)
30 # a diákok átlagéletkora
31 print("\n----- A diákok átlagéletkora: -----")
32 osszeg=0
33 for diak in diakok:
34     osszeg += diak['kor']
35 atlag=osszeg / len(diakok)
36 print("A hallagatók átlagéletkora: %.2f év." % (atlag))
37 # a legidősebb diák nevének és korának kiíratása
38 print("\n----- A legidősebb diák: -----")
39 max_index=0
40 max_kor=diakok[0]['kor']
41 for index, diak in enumerate(diakok):
42     if diak['kor'] > max_kor:
43         max_kor = diak['kor']
44         max_index = index
45 print("A legidősebb hallgató kora %d év" % (max_kor))
46 print("A legidősebb hallgató neve: ",diakok[max_index]['vezeteknev'],diakok[max_index]['keresztnev'])
```

- Ha beolvastuk az adatokat, feltöltöttük a listát a megfelelő típusú adatokkal, akkor végezzünk el pár alapvető feladatot!
- A 21-25. sortól **kiíratjuk a matematika szakos hallgatók neveit**. Megjegyzésbe írjuk le, hogy mit fogunk csinálni a faladatrészben (21. sor), majd a minta szerint egy print() utasítással elválasztjuk a feladatokat egymástól (22. sor). (A többi feladatnál is ezt használjuk.)
- Ha a diakok listában for utasítással végig megyünk (23. sor) és a szakok kulcsnál megegyeznek a ’matematika’ szóval (24. sor), ott írja ki a vezetékneveket és a keresztneveket, szóközzel elválasztva (összefűzve)(25. sor).

- A következő feladatrészben a **történelem szakos hallgatók összes adatát gyűjtjük ki**, és tesszük egy külön listába. (26-29. sor)
- Ezt a műveletet **leképezéssel készítjük** el. (28. sor)

```
28      tortenelem_szak=[diak for diak in diakok if diak['szak'] == 'történelem']
```

- Hozzunk létre egy **új listát „tortenelem_szak" néven**! A lista bejárását írjuk be először (for diak in diakok), majd megadjuk a feltételt, ami alapján szűrjük a diak szótárból a 'szak' kulcsnál megtalálható 'történelem' egyezést (diak['szak'] == 'történekelm'). A műveletnél változatlanul hagyjuk az értéket.
- A feladatrész végén **kiíratjuk** a „történekem_szak" lista tartalmát!
- A következő feladatrészben írassuk ki a **hallgatók átlagéletkorát**! (30-36. sor)
- Ez egy nagyon egyszerű feladat, hiszen már nagyon sokszor számoltunk átlagot. Létrehozunk egy „osszeg" nevű változót, melyben összeadjuk a „diakok" **lista bejárásával** a „diak" szótár 'kor' kulcsához tartozó értékeket. (33-34. sor)
- Majd egy „atlag" nevű változóba beletesszük az összeg és a „diakok" adatit tartalmazó lista hosszának hányadosát. Ahol a hányadost a len() függvénnyel kapunk meg. (35. sor)
- Végül formázottan kiírjuk az eredményt a minta szerint, ahol float adattípust használunk két tizedessel. (36. sor)
- Az utolsó részfeladatban **írassuk ki a legidősebb hallgató nevét és életkorát**! (37-45. sor)
- A részfeladat elején két változót adunk meg, amire szükségünk van. Az egyik a legnagyobb szám indexének eltárolásához kell majd, ezért a neve „max_index" lesz és az elején lenullázzuk. (39. sor)
- A másik változó a legnagyobb kor tárolására fog szolgálni. A kezdő értéknek pedig a „diakok" listánk nulladik elemének 'kor' kulcsához tartozó értéket adjuk meg. (40. sor)
- A következő sorban egy új beépített függvényt fogunk használni. A függvény neve: enumerate(). Az enumerate függvény segítségével számlálót használhatunk, amikor objektumokkal dolgozunk. Tegyük fel, hogy végig akarunk menni minden elemen a listánkban egy for ciklus segítségével. Emellett minden elem indexét is szeretnénk megtudni az iteráció során.
  Ezt megtehetjük például a range használatával. Ekkor le kell írnunk, hogy for x in range, majd használnunk kell a len-t, ami megmutatja a lista hosszát. A kettőspont után írhatunk egy print utasítást, ami az x-et és az x indexét is kinyomtatja. Eddig mindig ezt használtuk.
  Az enumerate-et for ciklussal együtt is lehet használni. Az enumerate függvény egy számlálót ad a listánkhoz, és a kimenet egy tuple lesz indexszámokkal a megfelelő elemek mellett.

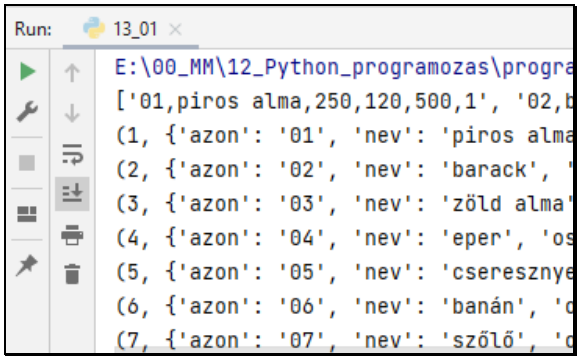
```
41      for index, diak in enumerate(diakok):
```

- A következőkben pedig, ahogy végig megyünk, megvizsgáljuk, hogy az aktuálisan tárolt legnagyobb érték (max_kor) nagyobb-e mint a szótár aktuális 'kor' kulcsához tartozó érték. (42. sor)
- Mert ha nagyobb akkor beletesszük az új értéket és ezzel megyünk tovább. (43. sor)
- És ha a vizsgálat két sorral előbb igazzá vált szükség lesz a legnagyobb kor indexére, hogy a nevet is ki tudjuk íratni.
- Végül a minta szerint formázottan kiíratjuk a legidősebb hallgató korát és nevét.

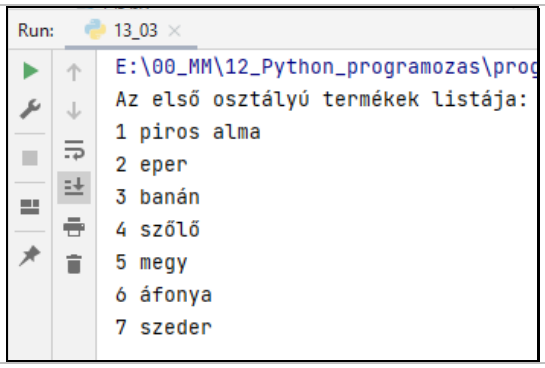
```
Run:  19e
E:\00_MM\12_Python_programozas\programok\venv\Scripts\python.exe E:/00_MM/12_Python_prograi

----- Matematika szakos hallagatók listája: -----
Nagy Béla
Tóth Imre
Fekete Ivett

----- Történelem szakos hallagatók adatai: -----
[{'vezeteknev': 'Kiss', 'keresztnev': 'János', 'kor': 20, 'szak': 'történelem', 'kollegist

----- A diákok átlagéletkora: -----
A hallagatók átlagéletkora: 20.40 év.

----- A legidősebb diák: -----
A legidősebb hallgató kora 23 év
A legidősebb hallgató neve:  Farkas Zsanett
```

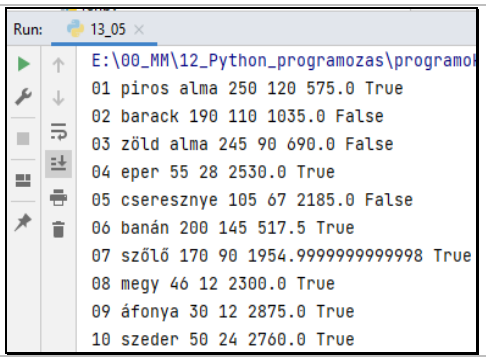
GYAKORLATI FELADATOK (19. témakörhöz)

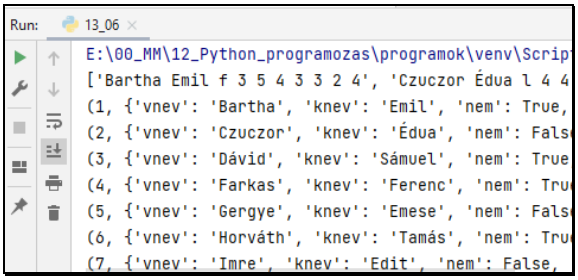
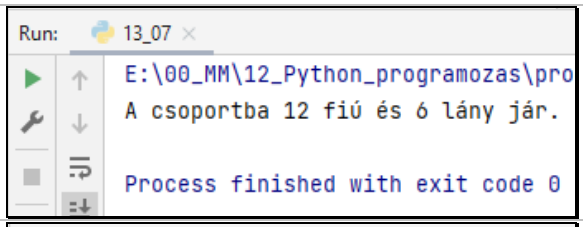
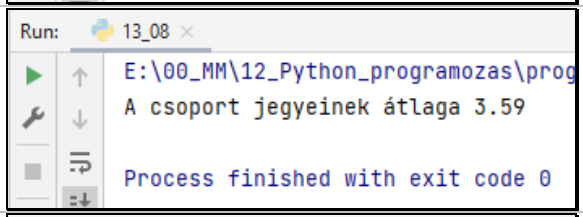
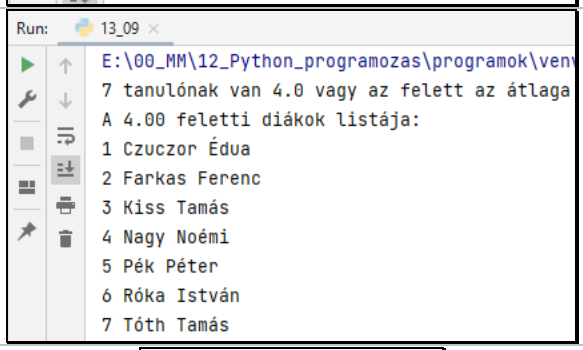
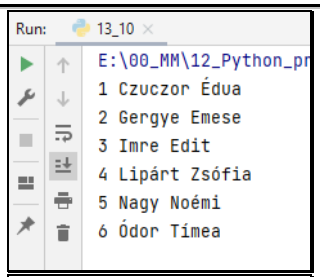
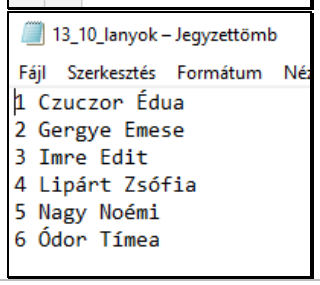
FELADAT LEÍRÁSA	KÉPERNYŐKÉP

1. (13_01.py)
Egy zöldség-gyümölcs üzlet eltárolt bizonyos adatokat egy txt fájlba. A fájlban 6 mező és 10 rekord van. A mezők sorban: azonosító, gyümölcs neve, eredeti készlet /kg, eladott mennyiség /kg, 1 kg gyümölcs ára, és hogy első osztályu-e a termék.
Először olvastasd be egy „**gyumolcsok**” nevű listába az adatokat, majd írasd ki a képernyőre egy sorba!
Majd készíts szótár típust a következő kulcsokkal „**termek**” néven: azon, nev, ossz_kg, eladott_kg, kg_ar, a_oszt. A kg-ok legyenek integer típusúak, az „a_oszt” legyen logikai típus!
Ezeket tedd bele egy „**termekek**” listába!
Végül írasd ki a képernyőre a termékeket, külön sorban a minta szerint! Használd az enumerate() függvényt!
`for i in enumerate(termekek, start=1):`
`  print(i)`
	

2. (13_02.py) Mentsd el az előző programot másnéven! Majd töröld ki a kiíratásos (print()) részeket! Aztán a minta szerint írasd ki a képernyőre valamelyik gyümölcs kilogrammonkénti árát! A kódot úgy írd meg, hogy ha másik gyümölcs árát akarod kiíratni, akkor csak egy helyen kelljen átírni a gyümölcs nevét és úgy is hibátlanul működjön!	

3. (13_03.py)
Mentsd el az előző programot másnéven! Majd töröld az előző feladat miatt begépelt sorokat!
Gyűjtsed ki egy „kigyujt” nevű listába az első osztályú gyümölcsök nevét!
Majd a minta szerit kistázd ki több sorba a gyümölcsök nevét! Használd az enumerate() függvényt
`for i,j in enumerate(kigyujt, start=1):`
`  print(i,j)`
	

4. (13_04.py) Mentsd el az előző programot másnéven! Majd töröld az előző feladat miatt begépelt sorokat! Számold ki, és írasd ki a képernyőre, hogy mennyi volt a bevétel az eladott gyümölcsökből!	
5. (13_05.py) Mentsd el az előző programot másnéven! Majd töröld az előző feladat miatt begépelt sorokat! Készíts olyan programot, amelyben a beolvasás és szótár kialakítása után a gyümölcsök kilogrammonként árát megemeled 15%-al, aztán a megváltoztatott adatokkal kiíratod egy új „13_05_emelt_ar.txt” fájlba és a képernyőre a teljes „adatbázist” a minta szerint!	

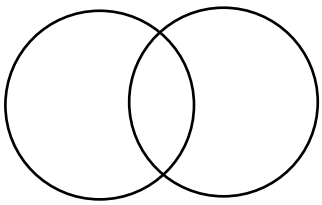
FELADAT LEÍRÁSA	KÉPERNYŐKÉP
6. (13_06.py) Egy iskola 9.A osztályának, matematika csoportjában tanulók adatait és érdemjegyeit tároljuk a 13_06_naplo.txt fájlban! A fájlban a tanulók nevét, nemét, és 7 db érdemjegyét találod. Először olvastasd be egy „**naplo**” nevű listába az adatokat, majd írasd ki a képernyőre egy sorba! Majd készíts szótár típust a következő kulcsokkal „**diak**” néven: vnev, knev, nem, jegy_1, … jegy_7. A „nem” legyen logikai típus Az érdemjegyek legyenek integer típusúak! Ezeket tedd bele egy „**diakok**” nevű listába! Végül írasd ki a képernyőre a diákokat, külön sorban a minta szerint! Használd az enumerate() függvényt!	
7. (13_7.py) Mentsd el másként az előző programot! Majd a minta szerint írasd ki (számold meg), hogy hány fiú és hány lány jár a csoportba!	
8. (13_08.py) Mentsd el az előző programot másnéven! Majd töröld az előző feladat miatt begépelt sorokat! Számold ki a diákok jegyeiből a csoport teljes átlagát, majd írasd ki a képernyőre a minta szerint!	
9. (13_09.py) Mentsd el az előző programot másnéven. Majd változtasd meg, egészítsed ki az előző programot úgy, hogy a mintán látható módon először kiírja a program, hogy hány olyan diák van, akinek 4.0, vagy az felett van az átlaga. Aztán listázd ki azok nevét, akik megfelelnek a feltételnek!	
10. (13_10.py) Mentsd el az előző programot másnéven. Majd változtasd meg, egészítsed ki az előző programot úgy, hogy a mintán látható módon a képernyőre kigyűjtse a lányok nevét! Majd ugyanezen lányokat írasd ki egy 13_10_lanyok.txt nevű fájlba is!	 

20.) HALMAZ TÍPUS (SET)

Az eddig megtanult, alkalmazott adattípusok között voltak az egyszerű típusok:

- int (egész számok)
- float (tizedes törtek)
- str (sztringek)
- bool (logikai típus: True / False)

Aztán tanultuk az összetett adattípusokat, melyekben nem csak egy értéket, hanem egymással logikailag összetartozó adatokat tudunk tárolni:



- lista (list)
- szótár (dict)

Ebben a leckében pedig egy új adattípusról lesz szó:

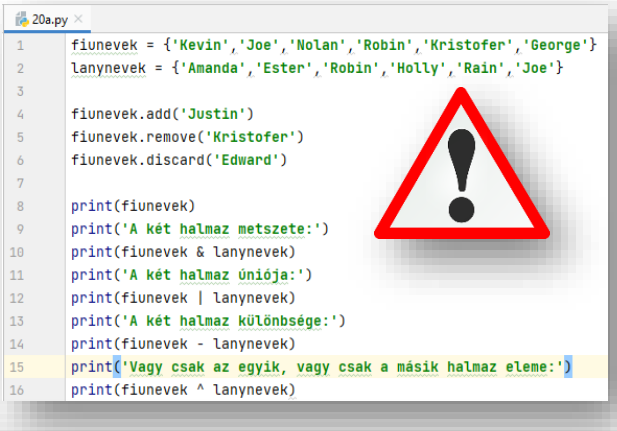
- **halmaz (set)**

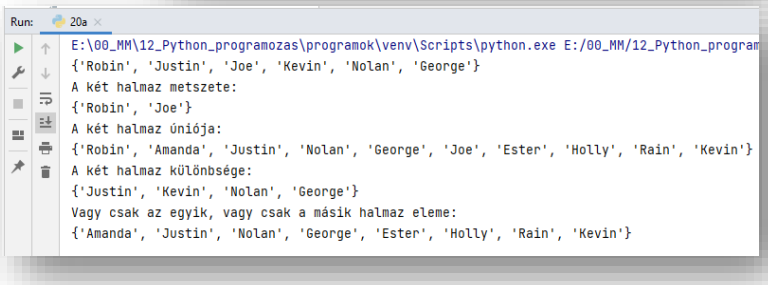
Nézzük meg a halamazok jellemzőit:

- a halmazokban **egy elem csak egyszer fordulhat elő** (a listákban megengedett volt az ismétlődés, itt nem)
- a **listák rendezetlen tárolók** (gondoljunk egy zsákra, belepakolunk dolgokat, majd tetszőleges sorrendben kivesszük belőle)
- **többféle adattípust tárolhatunk** a halmazokban (ebben hasonlít a listákra)
- az **elemeket nem lehet megváltoztatni** (a listákban például megduplázhattuk a tagok értékét, itt a halmazokban ezt nem lehet megtenni)
- a halmazokkal viszont a matematika órán megtanult műveletek elvégezhetőek (**metszet, unió, különbség**)
- sok hétköznapi probléma megoldásnál használhatjuk ezt a típust

Nézzünk példát a halmaz típus használatára:

(20a.py)

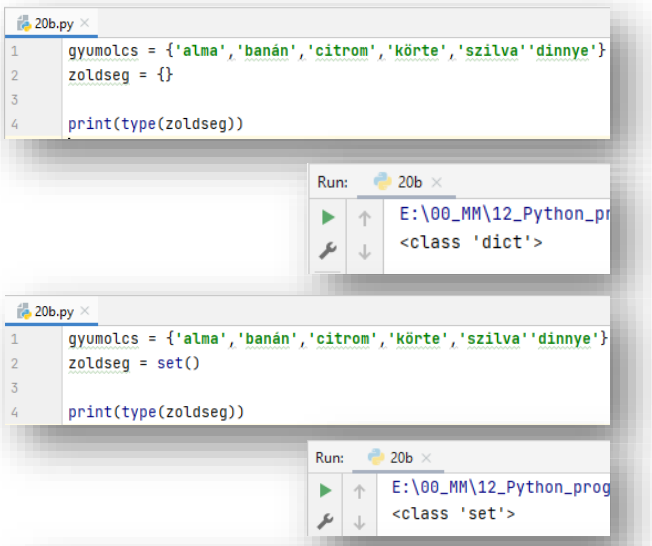




- A halmazok létrehozása, a név megadása után egyenlőség jel, majd kapcsoszárójelek között felsoroljuk a halmaz tagjait!
- A programban fiúneveket és lányneveket adunk meg.
- A halmazok bővítésére az „add()” metódust használjuk. egyszerűen a halmaz neve után ponttal beírjuk a parancsot, majd sima zárójelek között megadjuk azt a nevet, amivel bővíteni szeretnénk.
- Az eltávolításhoz használjuk a „remove()” metódust.
- Ezt csak akkor alkalmazhatjuk, ha biztos hogy a név szerepel a halmazban. Ha nem létezik, akkor hibát jelez.
- Van egy olyan metódus, amellyel ez a hibaüzenet elkerülhető, ha nem tagja a halmaznak a megadott érték. Ez pedig a „discard()” metódus.
- Aztán nézzük meg a minta szerint a két halmaz metszetét (&), unióját (|), különbségét (-), és azokat a taggokat, amelyek vagy csak az egyik, vagy csak a másik halmazban szerepelnek (^)!

(20b.py)

- Az új programban hozzunk létre egy „gyumolcs” nevű halmazt, melyben soroljunk fel gyümölcs neveket!
- Majd akarjunk létrehozni egy „zoldsegek” nevű üres halamazt! Itt egy kis problémába ütközünk. Ha az egyszerű logika szerint haladunk akkor két kapcsos zárójelet írunk egymás után. Írassuk ki „type” függvénnyel, hogy milyen típust hoz létre a python! Látjuk, hogy szótár (’dict’) lesz. Tehát ez így nem jó.
- A megoldás az, hogy meghívjuk az úgynevezett konstruktorát „set()

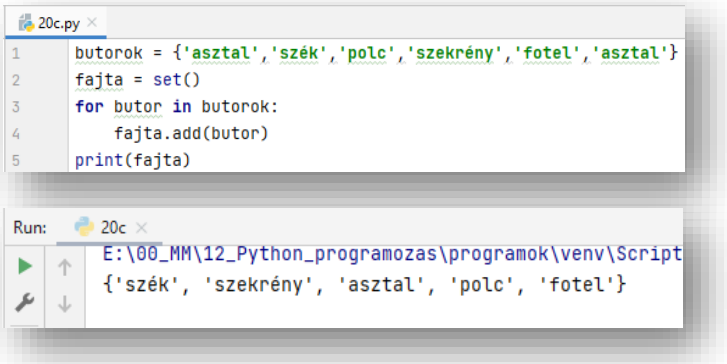


- De adhatunk meg a jobb oldalon látható módon halmazt, és elemeit!
- Egyébként látszik a kiíratásnál, a sorrend az nem ugyanaz, mint ahogyan én megadtam eredetileg.

- Ha létrehozunk egy olyan halmazt, melyben különböző típusú elemek vannak, azzal nincsen gond.
- Viszont, ha elemismétlés van, azt csak egyszer veszi figyelembe! Látjuk a kiíratásnál.

(20c.py)

Nézzünk egy olyan példát, ahol bútorok fajtáját soroljuk fel és ebben a halmazban vannak ismétlődések.
Sokszor fordul elő, hogy ezeket az ismétlődéseket kell megszüntetni ténylegesen.



- Tehát a program elején felsoroljuk egy „butorok” halmazban az elemeket.
- Majd létrehozunk egy üres „fajta” nevű halmazt, konstuktor segítségével.
- Aztán for ciklussal végig megyünk az elemeken (bejárjuk), és az add() metódussal beletesszük az üres halmazba az elemeket.
- Végül kiíratjuk az ismétlődésmentes halmazunkat.

21.) TUPLE TÍPUS

Most egy kevésbé használt típussal ismerkedünk meg. De vannak olyan feladatok, helyzetek, amikor kimondottan hasznos a **tuple típus** használata.
A legegyszerűbb megérteni a lényegét, ha a listából indulunk ki.

Lista:	**Tuple:**
rendezett, indexelet	**rendezett, indexelet**
egy elem többször szerepelhet	**egy elem többször szerepelhet**
többféle típust tárolhat	**többféle típust tárolhat**
meg lehet változtatni az elemek	**NEM LEHET MEGVÁLTOOZTATNI az elemek**
db számát, értékét, sorrendjét	**értékét, sorrendjét, számát**

Tehát a **listához képest kevesebb képességgel rendelkezik**, akkor miért alkalmazzuk? Hát pont ezért, **néha ez a cél, hogy ne tudjunk rajta változtatni**. Néha szükségünk lehet olyan tárolóra, melynek az értékén nem tudunk változtatni.

Hogy mikor célszerű alkalmazni? Amikor egy komolyabb programot készítünk és az egész feladat alatt szükségünk va állandó, megváltoztathatatlan értékekre. Például fix koordináták használatánál, vagy RGB színkódok alkalmazásánál, vagy éppen felbontások beállításánál.
Nézzünk példákat:

21a.py

```
1  koordinata_1 = (1,2)
2  koordinata_2 = (5,4)
3  print(koordinata_1,koordinata_2, sep='\n')
4  print(type(koordinata_1),type(koordinata_2),sep='\n')
5  print('____________________________')
6  koordinata_3 = -3,-6
7  koordinata_4 = 2,-7
8  print(koordinata_3,koordinata_4, sep='\n')
9  print(type(koordinata_3),type(koordinata_4),sep='\n')
10 print('____________________________')
11 x,y = koordinata_1
12 print('x = ',x, type(x))
13 print('y = ',y, type(y))
14 print('____________________________')
15 kulon_tip_tuple = ('nevek',3,6.7,True)
16 print(kulon_tip_tuple)
17 print(type(kulon_tip_tuple))
18 print(kulon_tip_tuple[1])
19 print('____________________________')
20 kulon_tip_tuple[3]=False
```

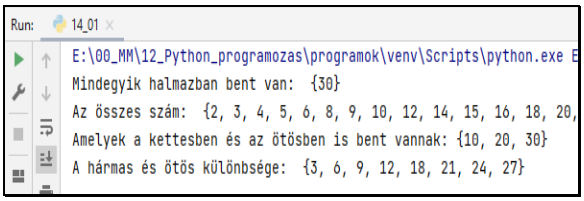
Run: 21a

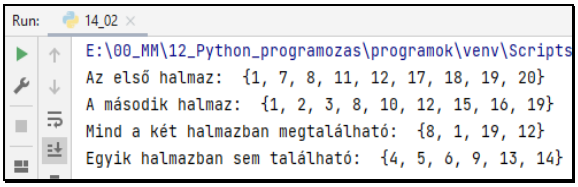
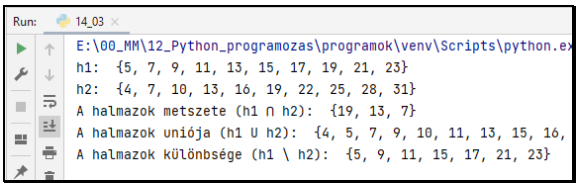
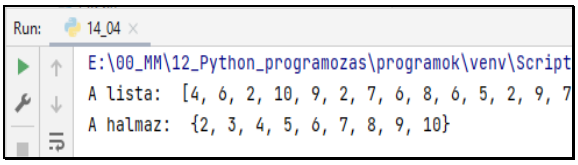
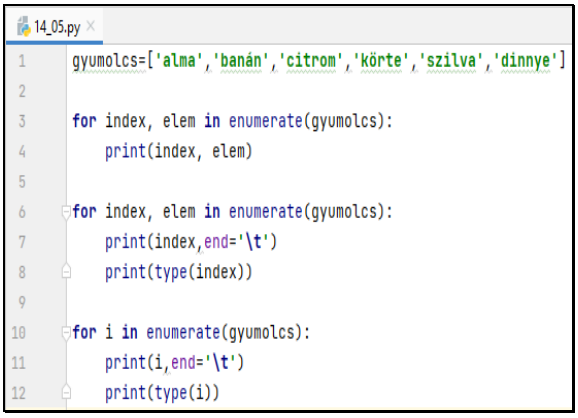
```
E:\00_MM\12_Python_programozas\programok\venv\Scripts\python.exe E:/00_MM/12_Py
(1, 2)
(5, 4)
<class 'tuple'>
<class 'tuple'>
--------------------------
(-3, -6)
(2, -7)
<class 'tuple'>
<class 'tuple'>
--------------------------
x =  1 <class 'int'>
y =  2 <class 'int'>
--------------------------
('nevek', 3, 6.7, True)
<class 'tuple'>
3
--------------------------
Traceback (most recent call last):
  File "E:\00_MM\12_Python_programozas\programok\21a.py", line 20, in <module>
    kulon_tip_tuple[3]=False
TypeError: 'tuple' object does not support item assignment
```

- Képzeljünk el egy koordináta tengelyt, melyen megadunk két pontot. (koordinata_1, és koordinata_2) Úgy tudunk megadni egy tuple típust, hogy a név megadása után egyenlőségjelet teszünk, majd sima zárójelek között megadjuk az értékeket, vesszővel elválasztva. (1-2. sor)
- Írassuk ki egymás alá, majd nézzük meg a típusukat! (3-4. sor)
- Adjunk meg még két koordinátát az előzőek szerint, de most zárójelek nélkül! Látjuk, hogy így is működik. (5-6. sor)
- Nézzünk példát a tuple adattípus kicsomagolására!
- Adjunk a 11. sorban az x,y –nak a koordinata_1-et, és ha külön-külön sorban kiíratjuk az elemeket és megnézzük a típusait, láthatjuk, hogy „int” típusokat kapunk. (12-13. sor)
- Aztán nézzünk különböző típusú elemekkel feltöltött tuple-t! (15. sor)
- Írassuk ki a tuple-t (16. sor), majd nézzük meg a típusát (17. sor)
- Ahogy az elején megmondtuk, indexelhető is a tuple. Írassuk ki az egyik értéket! (18. sor)
- Végül próbáljuk meg felülírni a 3. indexű elemet! (20. sor)
- Futtassuk a programot, és látni fogjuk, hogy hibát jelez, hiszen a tuple típust nem lehet felülírni.

GYAKORLÓ FELADATOK (20-21. témakörökhöz)

FELADAT LEÍRÁSA	KÉPERNYŐKÉP

1. (14_01.py)
Készíts programot, melyben a következő halmazokat hozod létre az elején:
`kettes = {2,4,6,8,10,12,14,16,18,20,22,24,26,28,30}`
`harmas = {3,6,9,12,15,18,21,24,27,30}`
`otos = {5,10,15,20,25,30}`
Majd válaszolj a képernyőképen látható kérdésekre a minta alapján!
	

2. (14_02.py) Készíts programot, melyben az elején létrehozol négy üres halmazt! (h1, h2, h3, h4) Majd az első kettőbe 10-10 darab véletlen számot generálsz 1 és 20 között! (h1, h2) Aztán kiíratod a minta szerint azokat a számokat, amelyek mind a két halmazban megtalálhatóak! Ezt beleteszed a harmadik halmazba! (h3) Végül a feladat legnehezebb része kiíratni azokat a számokat, melyek nincsenek bent sem a h1-ben, sem a h2-ben. Ehhez segítségként: a h4-be 1-től 20-ig beletesszük az összes számot, majd azzal dolgozunk tovább!	
3. (14_03.py) Adott két függvény (y=2x+3 és y=3x+1), mindkettő értelmezési tartománya az egész számok [0;10] intervallumon. A program készítsen egy-egy halmazt a függvények értékkészleteivel, írja ki ezeket a képernyőre, majd jelenítse meg a halmazok metszetét, unióját és különbségét!	
4. (14_04.py) Készítsél programot, melyben létrehozol egy listát, „lista” néven, melyben felveszel 20 darab véletlen számot 1 és 10 között! Majd halmazt készítesz belőle, „halmaz” néven. (Magyarul minden szám csak egyszer szerepelhet a halmazban.)	
5. (14_05.py) Most kivételesen egy fordított feladatot látsz. Itt a kódot kell begépelned a mint alapján és futtatáskor kell értelmezni! Először létrehozol egy listát! Majd „enumerate” utasítással kiíratod a képernyőre az elemeit! Aztán az index számát és típusát, tabulátorral elválasztva! Ez azért fontos, mert itt ki van csomagolva! (index,elem) Azutolsó résznél pedig egy „i” változót használunk kicsomagolás nélkül, ezért a típusa tuple lesz.	

22.) PROGRAMOZÁSI TÉTELEK ALKALMAZÁSA A PYTHONBAN

Ebben a fejezetben fogunk olyan dolgokkal találkozni, amelyeket már tanultunk, ezért felfoghatjuk ismétlésnek. És természetesen elő fog fordulni új anyag is.
Ebben a tananyagban az úgynevezett „programozási tételek”-et nézzük át!
Melyek az:

• Összegzés	• Kiválasztás	• Szélsőérték meghatározása
• Megszámolás	• Másolás	• Metszet
• Eldöntés	• Kiválogatás	• Unió
• Keresés	• Szétválogatás	• Rendezések

Tehát a következő „programozási tételek” a feladatok során legtöbbször előforduló algoritmusokat tartalmazza. Mindegyik „tételhez” tartozni fog egy rövid magyarázat, az általános mondatszerű leírás egy szövegdobozban, és maga a python példa kódja és képernyőképe. Legtöbbször látjuk majd, hogy a python-ban sokkal egyszerűbben meg tudjuk oldani a kódolást.
A feladatok megoldásánál (mondatszerű leírásnál és a kódírásnál is) tömböket fogunk használni, a t[ ] tömbnek pedig n darab eleme lesz. A tömbök elemeinek indexe 0-tól n-1-ig fog menni. Ahol több tömbbel dolgozunk ott az első tömb az a[ ], amelynek n eleme van, a második tömb a b[ ], amelynek m elem van. Harmadik tömb neve pedig a c[ ]. Az a[ ] tömb ciklus változója szokásosan i, a b[ ] tömbbé j, a harmadik c[ ] tömbbé k. De ahol szükséges használjuk az x,y,z változókat is.
A mondatszerű leírásnál is a tömbök indexelését 0-val kezdjük. Az értékadást egy darab egyenlőségjel (=) jelenti, az egyenlőség vizsgálatot két egyenlőségjel (==) jelzi, a nem egyenlőt egy kisebb és egy nagyobb jel jelzi (<>).
A python program kód írásánál a tömbökbe véletlen számokat fogunk generálni és azokkal fogunk dolgozni.

(22a.py)
Összegzés
Magyarázat:
Az egyik legegyszerűbb „programozási tétel” az összegzés.
Adott egy 20 elemű tömb, melyet feltöltünk 1-20 –ig vél. számokkal.
Majd egy for ciklussal végig megyünk a tömb elemein és egy „osszeg” nevű változóba mindig hozzáadjuk az aktuális értéket.
Tehát egy **tömb elemértékeinek összegét** számoljuk ki.
Mondatszerű leírás:

```
osszeg = 0
ciklus i = 0 .. n -1
    osszeg = osszeg + t[i]
ciklus vége
ki osszeg
```

```
22a.py
1  from random import *
2  t=[]
3  for i in range(1,21):
4      t.append(randrange(20)+1)
5  print(t)
6  osszeg=0
7  for i in t:
8      osszeg=osszeg+i
9  print(osszeg)
```

```
Run:  22a
E:\00_MM\12_Python_programozas\programok
[18, 8, 6, 14, 12, 11, 13, 8, 3, 18, 11,
197
```

(22b.py)
Megszámolás
Magyarázat:
Adott feltételek alapján a **tömb bizonyos elemeit megszámoljuk**.
Pl.: Megszámoljuk mennyi negatív és pozitív szám van a tömbben.
Létrehozunk egy tömböt, amit feltöltünk -10-től +10-ig számokkal.
Majd indítunk egy for ciklust, végig a tömbön, ahol egy if feltétellel megvizsgáljuk hogy az adott érték kisebb-e mint 0. Mert ha kisebb akkor a „negativ” változó értékét egyel növeljük.
Különben a „pozitiv” változó értékét növeljük egyel.
Végül kiíratjuk a minta szerint.

Mondatszerű leírás:

```
szamlalo = 0
ciklus i = 0 .. n - 1
   ha t[i] < 0 akkor
      szamlalo = szamlalo + 1
   ha vége
ciklus vége
ki szamlalo
```

```
Run:  22b
E:\00_MM\12_Python_programozas\programok\venv\Scripts\python.exe E:/00_MM
[1, -8, -10, 4, -8, -2, 1, -10, 5, 0, 8, 3, -10, -4, -8, 2, 10, 0, 9, 9]
A tömbben 8 negatív és 12 pozitív szám van
```

A megszámolás python kódja:

```python
from random import *
t=[]
negativ=0
pozitiv=0
for i in range(1,21):
    t.append(randrange(21)-10)
print(t)
for i in t:
    if i<0:
        negativ += 1
    else:
        pozitiv += 1
print('A tömbben %d negatív és %d pozitív szám van' % (negativ,pozitiv))
```

(22c.py)

Eldöntés

Magyarázat: *Mondatszerű leírás:*

Az eldöntés esetében azt vizsgáljuk, hogy **szerepel-e egy bizonyos tulajdonságú elem az adatsorban**. A válasz igen/nem (True vagy False lehet.)

Létrehozunk egy tömböt. Amit feltöltünk 1-20-ig terjedő véletlen számokkal. Majd „talalt” néven létrehozunk egy változót, amit „False”-ra állítunk.

Aztán indítunk egy „while” ciklust, ami mindaddig fut, amig nem talál egy olyan számot, amit ha elosztunk hárommal nullát nem kapunk eredményül.

Mert, ha talált ilyet, akkor a „találat” változót igazra állítjuk. Közben egy „i” változót felhasználva léptetjük a tömb indexét. Ami egyébként fontos lesz a keresésnél.

A listát bejárhattuk volna egy for ciklus segítségével, de felesleges végighaladnunk az összes elemen.

A fenti program amint megtatálja az első olyan elemet, amely megfelel a feltételeknek, befejezi a lista átvizsgálását.

Hiszen ha már találtunk egy ilyen elemet, megvan a válasz.

```
van = 0
ciklus i = 0 .. n-1
  ha tomb[i] = ker_ertek akkor
     van = 1
  ha vége
ciklus vége
ki van
```

```python
from random import *
t=[]
for i in range(1,11):
    t.append(randrange(21))
print(t)
talalat = False
i=0
while i < len(t) and not talalat:
    if t[i] % 3 == 0:
        talalat = True
    i= i + 1
if talalat:
    print('Van a listában hárommal osztható szám.')
else:
    print('Nincs a listában hárommal osztható szám.')
```

```
Run: 22c
E:\00_MM\12_Python_programozas\programo
[19, 7, 7, 12, 18, 16, 8, 18, 14, 8]
Van a listában hárommal osztható szám.
```

(22d.py)

Keresés

Magyarázat: *Mondatszerű leírás:*

Most ugyanazt vizsgáljuk, mint az előbb, csak még azt is keressük, hogy a **megtalált elem hányadik helyen van**.

Annyi dolgunk van, hogy elmentjük az előző programot másként és kiegészítjük egyetlen sorral. Mivel az előbb az „i” változóval lépkedünk előre egyesével, itt csak fel kell használni a kiíratásnál.

```
van = 0
ciklus i = 0 .. n-1
  ha tomb[i] = ker_ertek akkor
     van = 1
  ha vége
  i = i+1
ciklus vége
ki tomb[i]
```

```python
if talalat:
    print('Van a listában hárommal osztható szám.')
    print('Az első szám a listából, ami osztható hárommal: ',t[i-1])
```

```
Run: 22d
E:\00_MM\12_Python_programozas\programok\venv\Script
[9, 18, 9, 16, 1, 16, 11, 5, 18, 13]
Van a listában hárommal osztható szám.
Az első szám a listából, ami osztható hárommal:  9
```

(22e.py)

Kiválasztás

Magyarázat:

A kiválasztás tételt akkor használjuk, ha **tudjuk, hogy a keresett értéket tartalmazza a tömb**. Ezért azt nem vizsgáljuk, hogy vége van-e a tömbnek. Ezért most fixen feltöltünk 1 és 20 közötti számokkal egy „t” tömböt, majd addig megyünk egy ciklussal az elemeken végig, amíg meg nem találja.
Végül kiíratjuk, hogy **hányadik elem**.

```
t=[7,15,3,6,11,18,14,7,4,9,12,13,17]
i=0
while t[i] != 12:
    i += 1
print('A 12-es szám a(z) %d -dik elem a lisában.' % (i+1))
```

```
Run: 22e
E:\00_MM\12_Python_programozas\programok\ve
A 12-es szám a(z) 11 -dik elem a lisában.
```

Mondatszerű leírás:

```
i = 0
ciklus amíg tomb[i] <> keresett
   i = i + 1
ciklus vége
ki i + 1
```

(22f.py)

Másolás

Magyarázat:

Ennél a hasznos „programozási tétel”-nél **egy sorozat elemeit átmásolom egy másik sorozatba, miközben valamilyen átalakítást végzek az egyes elemeken**.
Ebben a példában egy „a” nevű tömbben felsorolt számokat duplázzuk meg egy „b” nevű listába.

Mondatszerű leírás:

```
ciklus i = 1 .. n
   b[i] = művelet(a[i])
ciklus vége
```

```
a=[7,15,3,6,11,18,14,7,4,9,12,13,17]
b=[]
i=0
for i in range(len(a)):
    b.append(a[i]*2)
print(a)
print(b)
```

```
Run: 22f
E:\00_MM\12_Python_programozas\programok\venv\Scrip
[7, 15, 3, 6, 11, 18, 14, 7, 4, 9, 12, 13, 17]
[14, 30, 6, 12, 22, 36, 28, 14, 8, 18, 24, 26, 34]
```

(22g.py)

Kiválogatás

Magyarázat:

Az egyik tömb elemeit **átteszem egy másik tömbbe, ha megfelelnek egy adott feltételnek**. Tehát kiválogatom valamilyen feltétel szerint.
Adott a és b tömb. Az a tömb egész számokat tartalmaz 1 és 20 között. Az a tömbből a 10, vagy annál kisebb számokat átrakom b tömbbe.

Mondatszerű leírás:

```
j = 0
ciklus i = 0 .. n - 1
   ha a[i] <= 10
      b[j] = a[i]
      j = j + 1
   ha vége
ciklus vége
```

```
a=[7,15,3,6,11,18,14,7,4,9,12,13,17]
b=[]
i=0
for i in range(len(a)):
    if a[i] <= 10:
        b.append(a[i])
print(a)
print(b)
```

```
Run: 22g
E:\00_MM\12_Python_programozas\programok\venv\S
[7, 15, 3, 6, 11, 18, 14, 7, 4, 9, 12, 13, 17]
[7, 3, 6, 7, 4, 9]
```

(22h.py)

Szétválogatás

Mondatszerű leírás:

Magyarázat:

Két tömbbe válogatjuk szét egy tömb elemeit. Adott „a” tömb, amely egész számokat tartalmaz 1 és 20 között. A „b” és „c” tömb pedig üres. Az „a” elemeit „b” tömbbe rakjuk ha egyenlők vagy kisebbek 10-él, különben c-ben tároljuk.

```
j = 0
k = 0
ciklus i = 0 .. n-1
   ha a[i] < 5
      b[j] = a[i]
      j = j + 1
   különben
      c[k] = a[i]
      k = k + 1
   ha vége
ciklus vége
```

```python
a=[7,15,3,6,11,18,14,7,4,9,12,13,17]
b=[]
c=[]
i=0
for i in range(len(a)):
    if a[i] <= 10:
        b.append(a[i])
    else:
        c.append(a[i])
print(a)
print(b)
print(c)
```

```
E:\00_MM\12_Python_programozas\programok\venv\Sc
[7, 15, 3, 6, 11, 18, 14, 7, 4, 9, 12, 13, 17]
[7, 3, 6, 7, 4, 9]
[15, 11, 18, 14, 12, 13, 17]
```

(22i.py)

Szélsőérték meghatározása

Mondatszerű leírás:

```
min=t[0]
ciklus i = 1 .. n-1
   ha t[i] < min akkor
      min = t[i]
   ha vége
ciklus vége
ki min
```

```
max = t[0]
ciklus i = 1 .. n - 1
   ha t[i]> max akkor
      max = t[i]
   ha vége
ciklus vége
ki max
```

Magyarázat:

Tulajdonképpen ez a **legkisebb és/vagy legnagyobb érték kiválasztása egy listából**.

A példában generáljuk 10 db véletlen számot 1 és 100 között.

Majd keressük ki a minimum és a maximum értékeket a tömbből.

Ennél a „tételnél” arra kell figyelni, hogy a program elején a minimum értéket az előfordulható legnagyobbra, a maximum értéket az előfordulható legkisebbre kell állítani, mert az első összehasonlításnál már cserélődni kell az értékeknek (példa).

Vagy egy másik megoldásként a minimum és maximum értékeket a tömb első elemére állítjuk be (mondatszerű leírás).

```python
from random import *
t=[]
for i in range(1,11):
    t.append(randrange(100)+1)
print(t)
min=100
max=0
for i in range(len(t)):
    if t[i] < min:
        min = t[i]
for i in range(len(t)):
    if t[i] > max:
        max = t[i]
print('A legkisebb elem: ',min)
print('A legnagyobb elem: ',max)
```

```
E:\00_MM\12_Python_programozas\programok
[47, 25, 62, 71, 33, 35, 2, 10, 13, 81]
A legkisebb elem:  2
A legnagyobb elem:  81
```

(22j.py)

Metszet

Magyarázat: *Mondatszerű leírás:*

Két tömb azonos elemeinek kiválogatása egy harmadik tömbbe.

Nem olyan régen tanultuk a halmazokat. Ott két halm. metszetét az „print(a & b) paranccsal oldottuk meg. Itt ez nem működik, mert most egy külön listába szeretném eltárolni a közös értékeket.

Ezért két egymásba ágyazott ciklussal kell megoldani a feladatot.

Egyesével megyünk végig az „a” tömbön azon belül pedig megvizsgáljuk a „b” összes elemével, hogy van-e egyezés. Mert ha igen, akkor hozzáadjuk a „c” listához.

Értelmezzük a programot!

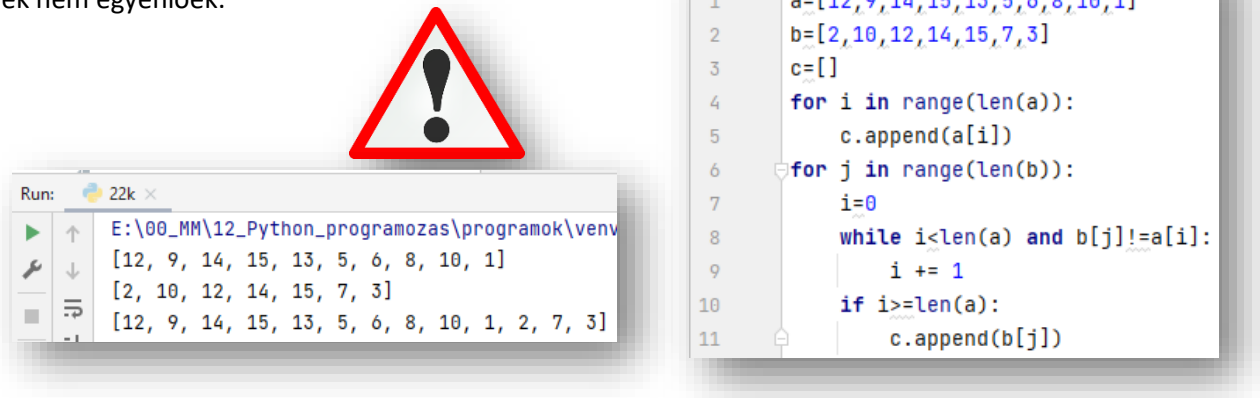
```
k = 0
ciklus i = 0 .. n-1
   j = 0
   ciklus amíg j<m és b[j]<>a[i]
      j = j + 1
   ciklus vége
   ha j<m akkor
      c[k] = a[i]
      k = k + 1
   ha vége
ciklus vége
```

```python
a=[12,9,7,15,13,5,8,11,19,3,1,10]
b=[2,4,6,8,10,12,14,16,17,18,20]
c=[]
i=0
for i in range(len(a)):
    j=0
    for j in range(len(b)):
        if a[i]==b[j]:
            c.append(a[i])
print(a)
print(b)
print(c)
```

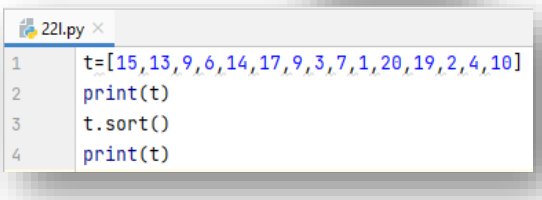
```
E:\00_MM\12_Python_programozas\programok\ve
[12, 9, 7, 15, 13, 5, 8, 11, 19, 3, 1, 10]
[2, 4, 6, 8, 10, 12, 14, 16, 17, 18, 20]
[12, 8, 10]
```

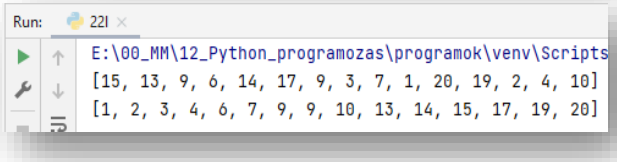
(22k.py)
Unió
Magyarázat: *Mondatszerű leírás:*
A és B tömb minden elemét szeretnénk C tömbbe tenni.
Először C-be tesszük az A összes elemét, majd ami nem szerepel A-ban, azt beletesszük C-be.
A mondatszerű leírás átalakítása python nyelvre mindig kihívás.
Először generálunk „a” és „b” tömböt 1 ás 20 közötti számokkal!
Létrehozunk egy üres „c” tömböt.
Aztán beletesszük egy for ciklussal az „a” tömb tartalmát a „c” tömbbe.
Uniónál arra kell figyelni, hogy metszet számai ne szerepeljenek kétszer a „c” tömbben.
Ezért a „b” tömb elemein egyesével végig megyünk egy hátultesztelő ciklussal és csak azokat a számokat adjuk hozzá a „c” tömbhöz, amelyek nem egyenlőek.

```
ciklus i = 0 .. n-1
   c[i] = a[i]
ciklus vége
k = n
ciklus j = 0 .. m-1
   i = 0
   ciklus amíg i<n és b[j]<>a[i]
      i = i + 1
   ciklus vége
   ha i>=n akkor
      c[k] = b[j]
      k = k + 1
   ha vége
ciklus vége
```



(22l.py)
Rendezések
Magyarázat:
A programozásban az egyik legfontosabb művelet a rendezés.
Amikor **egy rendezetlen tömb elemeit növekvő sorba rendezzük**.
Ennek a megoldására számtalan „programozási tétel” áll rendelkezésünkre.
A python nyelven ezt a problémát nagyon gyorsan és egyszerűen megoldhatjuk a short() metódussal.



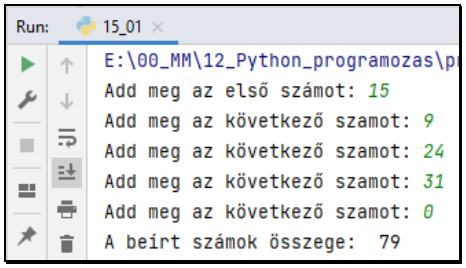
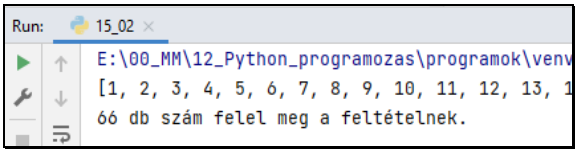
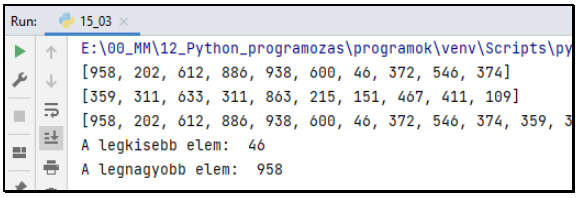
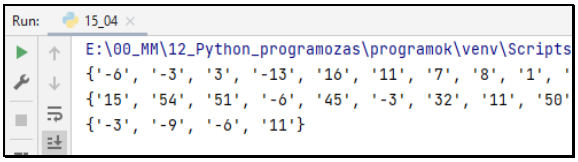
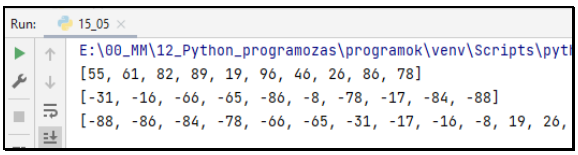


Ezek a különböző módszerek a gyorsaság, bonyolultság és memória igény függvényében változhatnak.
Példák a **rendezés típusai**ra:

- **Buborékrendezés**
- **Rendezés cserével**
- **Rendezés maximum kiválasztással**
- **Minimum kiválasztásos rendezés**
- **Rendezés beszúrással**
- **Shell rendezés**
- **Gyorsrendezés**
- **Összefésüléses rendezés**

Ezeket a rendezéses „tételeket”, és azok mondatszerű leírását érdemes megtanulni, megérteni, mert a későbbi tanulmányok során szükség lehet rájuk. Mind elméleti, mind gyakorlati téren. Az Interneten nagyon sok helyen utána lehet nézni.

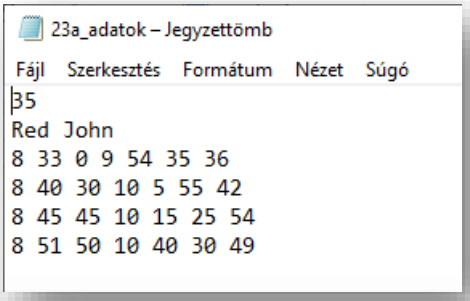
GYAKORLÓ FELADATOK (22. témakörhöz)

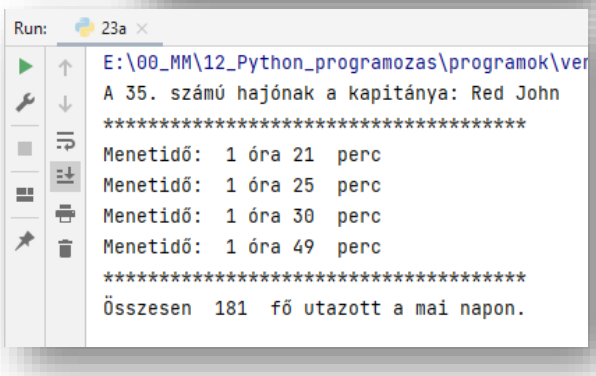
FELADAT LEÍRÁSA	KÉPERNYŐKÉP
1. (15_01.py) Készíts programot, amely 0 végjelig bekér számokat a felhasználótól, és a program végén kiírja az eddig begépelt számok összegét!	
2. (15_02.py) Hány darab olyan szám van ezerig, amely osztható hárommal is és öttel is? Készíts programot, melyben egy „t” nevű tömbbe beleteszed 1-1000-ig a számokat, majd végig mész a tömbön és megvizsgálod hány olyan szám van ami megfelel a feltételnek.	
3. (15_03.py) Hozzál létre három tömböt: paros, paratlan, szamok néven! A „paros” nevű tömbbe generálj 1 és 1000 közötti véletlen páros számokat! A „paratlan” nevű tömbbe generálj 1 és 1000 közötti véletlen páratlan számokat! Majd ezt a két tömböt add össze a „szamok” nevű tömbbe! Végül keresd ki a „szamok” tömbből a legkisebb és legnagyobb számot!	
4. (15_04.py) Készíts programot, melyben beolvasod a nyersanyag mappából a 15_04_a-txt és 15_04_b-txt fájlokat! Majd halmazokká alakítod őket! Aztán kiíratod a képernyőre és egy 15_04_metszet.txt nevű fájlba azokat a számokat, amelyek mind a két halmazban szerepelnek!	
5. (15_05.py) Készíts programot, melyben létrehozol három tömböt (a ,b, c)! Az „a” és a „b” tömböt feltöltöd 10-10 darab, -100 és 100 közötti véletlen számokkal. Írasd ki a két tömböt a képernyőre! Majd az „a” és „b” tömböt, összefésülöd” a „c” nevű tömbbe! (összeadod és rendezed) A „c” tömböt írasd ki a képernyőre és egy 15_05_sor,txt nevű fájlba is!	

KOMPLEX FELADATOK MEGOLDÁSA

(23a.py)

Készítsük el a következő összetetteb feladatot!
Van egy hajótársaság, akik kiránduló járatokat indítanak csütörtöktől vasárnapig, a tengeren az egyik városból a másikba. Van egy 23a_adatok.txt nyersanyag. Ebben a példában nem homogén adatok vannak a szövegfájlban. Az első sorban a hajójárat számát találjuk, a másodikban a kapitány nevét. Aztán a harmadik sortól viszont már szóközökkel elválasztott számokat találunk. Az első az az indulás órája, majd a perce és másodperce; a negyedik adat a sorban az az érkezés órája, perce és másodperce. Az utolsó adat a sorban az utazók száma!
Tehát a harmadik sorban a csütörtöki adatok, az utolsóban a vasárnapi adatok találhatóak.
A feladatunk az, hogy a minta szerint az elején kiírja (fájlból kiolvasva), hogy hányas számú hajónak, ki a kapitánya!
Majd, hogy melyik napon, mennyi volt a menetidő.
Végül, hogy hányan utaztak a hajóval ezen a héten!





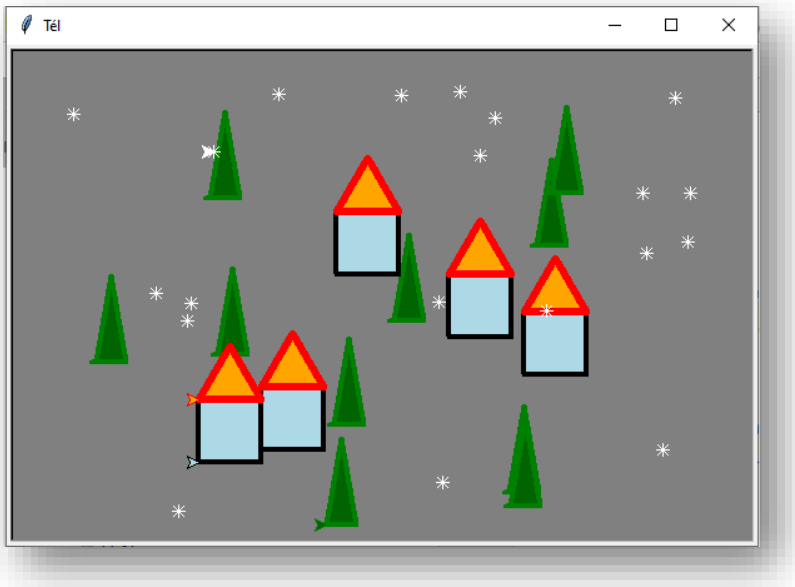
- A program elején létrehozunk egy „t” nevű üres tömböt. (1)
- A járat számát tároló „j_szam” változót, amit lenullázunk. (2)
- Egy „j_kap nevű változót, amit sztringre állítunk, mert ebben a kapitány nevét fogjuk tárolni. (3)
- Majd megnyitjuk olvasásra a nyersanyagot! (4)
- A következő két sorban beletesszük a megfelelő változókba a forrás első két sorát! Ehhez a readlin().strip() utasítást alkalmazzuk. (5-6)
- Aztán indítunk egy ciklust melyben soronként elhelyezzük az adatokat a „t” tömbbe, szóközök és sorvégi bekezdésjel nélkül. Erre a sor.strip().split() utasítást alkalmazzuk. (8)
- A következő 12 sorban létrehozunk egy szótár típust.
- Melyben először megadjuk a szótár nevét. (9).

```
23a.py
1  t=[]
2  j_szam=0
3  j_kap=str()
4  with open('./nyersanyag/23a_adatok.txt', 'r', encoding='utf-8') as forras:
5      j_szam=forras.readline().strip()
6      j_kap = forras.readline().strip()
7      for sor in forras:
8          t.append(sor.strip().split())
9  adat={}
10 adatok=[]
11 for elem in t:
12     hajo_adatok = elem
13     adat['ind_ora'] = int(hajo_adatok[0])
14     adat['ind_perc'] = int(hajo_adatok[1])
15     adat['ind_mp'] = int(hajo_adatok[2])
16     adat['erk_ora'] = int(hajo_adatok[3])
17     adat['erk_perc'] = int(hajo_adatok[4])
18     adat['erk_mp'] = int(hajo_adatok[5])
19     adat['szem_szam'] = int(hajo_adatok[6])
20     adatok.append(adat)
21     adat={}
22 print('A ',j_szam,'. számú hajónak a kapitánya: ',j_kap, sep='')
23 print('**************************************')
24 for adat in adatok:
25     print('Menetidő: ',int(abs((adat['ind_ora']*60+adat['ind_perc'])
26         -(adat['erk_ora']*60+adat['erk_perc']))/60),'óra',
27         int(abs((adat['ind_ora']*60+adat['ind_perc'])
28         -(adat['erk_ora']*60+adat['erk_perc']))%60),' perc')
29 print('**************************************')
30 ossz=0
31 for adat in adatok:
32     ossz=ossz+adat['szem_szam']
33 print('Összesen ',ossz,' fő utazott a mai napon.')
```

- Majd egy üres tömbre is szükségünk lesz ennek a neve „adatok” lesz. (10)
- Aztán elindítunk egy ciklust amiben megadjuk a címkéket, másnéven megnevezéseket és hozzárendeljük az értékeket. (11)
- A „t” tömbünkben lévő – most még - sorokban megyünk végig és bevezetünk egy ideiglenes változót „hajo_adatok” néven.
- Tehát az „elem”-ben a legelején a „t” tömb első sorát találjuk. Ami a txt harmadik sora és a számok vannak benne. (Most még szövegként.) (12)
- „Felcímkézzük” a különböző „mezőinket”, és számmá alakítjuk int() függvénnyel az indexelt adatainkat. Pl.: adat['ind_ora'] = int(hajo_adatok[0]) (13-19)
- Végül hozzáadjuk az adatok tömbünkhöz a feldolgozott rekordjainkat: adatok.append(adat); és üritjuk az adat{}-ot. (20-21)
- Az első alfeladatunkban kiíratjuk mondatszerűen, hogy melyik hajónak, ki a kapitánya. Ehhez a program elején elkészített két változót használjuk fel. A txt-nk első két sorát íratjuk ki tulajdonképpen. (22)
- Aztán egy elválasztó vonalat íratunk ki a minta szerint.
- A második alfeladatban azt kell kiszámolnunk, hogy mennyi volt a menetidő a négy napon. Ehhez a indulás órájára, percére, és az érkezés órájára és percére van szükségünk. A másodpercekre nincs is szükségünk. Tulajdonképpen egy hosszú képletet kell készítenünk. Lehetne különböző ideiglenes változókat is létrehozni, de mi válasszuk azt a módszert, hogy teljes képletet készítünk több sorban.
- Úgy gondolkodunk, hogy percekkel számolunk, ezért az órákat megszorozzuk 60-al és hozzáadjuk a perceket. Mind az indulásnál mind az érkezésnél. Majd kivonjuk az egyiket a másikból.
- Alkalmazom az abs() függvényt, hogy semmi képpen ne kapjunk negatív számokat.
- Tehát a képlet: abs((ind_óra*60+ind_perc)-(erk_ora*60+erk_perc)). Ennek az eredményét percben kapjuk meg. Viszont a kiíratáshoz vissza kell térnünk az óra és percre. Úgyhogy a kapott percet elosztjuk 60-al és az egész részét felhasználjuk órának, a maradék részét pedig percnek.
- És ezt mind mondatba formázzuk a minta szerint. (24-28)
- Ismét választóvonalat szúrunk be!
- A harmadik részfeladat nagyon könnyű, hiszen az a kérdés, hogy a négy nap alatt hányan utaztak összesen a hajóutakon. Ezt pedig a txt-nk utolsó oszlopában szerepló számok összeadásával kapjuk meg. Ez a szótár típusunk utolsó mezője a „szem_szam” adatainak összeadásából jön ki. (30-33)
- Teszteld, ellenőrizd, javítsad a programot írás közben!

(23b.py)

Ebben az összetett feladatban visszatérünk a grafikus felülethez. Ezen kívül az eljárások használatához. A példában a jobb oldalon látható „téli tájat” kell elkészítenünk.
Tehát, egy 600*400-as szürke felületre kell elhelyeznünk véletlenszerű helyre 10 db fenyőfát 5 db házat és 20 db hópelyhet!
A házak fekete keretű, világoskék kitöltésű négyzetből, és egy piros keretű, narancssárga kitöltésű háromszögből álljon!
A „fenyőfák”, zöld rajzolószínnel és sötétzöld kitöltőszínnel legyenek megrajzolva!
A hópelyhek fehérek legyenek!



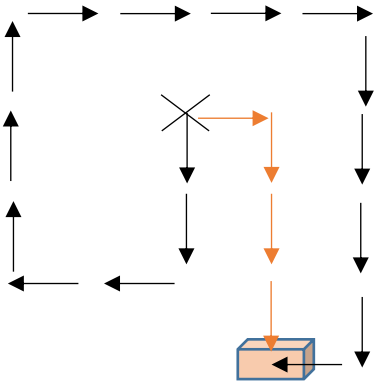
- A program elején beállítjuk az ablakunk tulajdonságait! Nevet adunk az ablaknak, megadjuk a háttérszínt, a címsor feliratát és az ablak méretét! (3-6.)
- Aztán négy „teknőst” állítunk be, az előző oldalon leírtak alapján, hogy a későbbiekben fel tudjuk használni őket. (8-25.)
- Majd elkészítjük az eljárásokat, amelyek megrajzolják a különböző alakzatokat! Ezek a négyzet, a háromszög, a hópihe, és a fenyőfa! (27-67.)
- Végül kirajzoltatjuk véletlen koordináták helyére először a 10 db fenyőfát, majd az 5 db házat, végül a 20 db hópehelyt! (69-83.)

```python
import turtle
from random import *
win = turtle.Screen()
win.bgcolor("gray")
win.title("Tél")
win.setup(600,400,0,0)

teki_1 = turtle.Turtle()
teki_1.color("black")
teki_1.pensize(4)
teki_1.fillcolor("lightblue")

teki_2 = turtle.Turtle()
teki_2.color("red")
teki_2.pensize(6)
teki_2.fillcolor("orange")

teki_3 = turtle.Turtle()
teki_3.color("white")
teki_3.pensize(1)

teki_4 = turtle.Turtle()
teki_4.color("green")
teki_4.pensize(4)
teki_4.fillcolor("darkgreen")

def negyzet(a,x,y):
    teki_1.up()
    teki_1.goto(x,y)
    teki_1.down()
    teki_1.begin_fill()
    for i in range(4):
        teki_1.forward(a)
        teki_1.left(90)
    teki_1.end_fill()

def haromszog(a,x,y):
    teki_2.up()
    teki_2.goto(x, y+a)
    teki_2.down()
    teki_2.begin_fill()
    for i in range(3):
        teki_2.forward(a)
        teki_2.left(120)
    teki_2.end_fill()

def ho(b,xb,yb):
    teki_3.up()
    teki_3.goto(xb,yb)
    teki_3.down()
    for i in range(8):
        teki_3.forward(b)
        teki_3.backward(b)
        teki_3.left(45)

def fa(z,xz,yz):
    teki_4.up()
    teki_4.goto(xz,yz)
    teki_4.down()
    teki_4.begin_fill()
    teki_4.forward(2*z)
    teki_4.left(100)
    teki_4.forward(5*z)
    teki_4.left(160)
    teki_4.forward(5*z)
    teki_4.left(100)
    teki_4.end_fill()

for i in range(10):
    xz=randrange(-280,240,1)
    yz = randrange(-180,100,1)
    fa(14,xz,yz)

for i in range(5):
    x=randrange(-200,200,10)
    y = randrange(-180,180,10)
    negyzet(50,x,y)
    haromszog(50,x,y)

for i in range(20):
    xb=randrange(-280,280,1)
    yb = randrange(-180,180,1)
    ho(5,xb,yb)

win.mainloop()
```

A feladatban a méretek és koordináták változhatnak. Az a lényeg, hogy a végeredmény hasonlítson az eredeti képernyőtervhez!
A program végén használjuk az ablak bezárására váró, mainloop() parancsot!

(23c.py)

A következő egyszerű példa a „kincskereső” feladat.
Melyben bekérjük karakterenként, hogy a kiindulási ponttól északra (E), délre (D), keletre (K), és nyugatra (N) mennyi egységet kell lépni! Majd a program megadja, egyszerűsítve a kincs helyét!



```
Run: 23c
E:\00_MM\12_Python_programozas\programok\venv\Sc
Kérem a robot parancsait: EEKEKEKKKNNDDDND
KK
Process finished with exit code 0
```

```
Run: 23c
E:\00_MM\12_Python_programozas\programok\ven
Kérem a robot parancsait: EEKNNNNDO
Nem megfelelő kód!
NNNE
Process finished with exit code 0
```

A képernyőkép mintája szerint vigyünk be kódokat!
Ha véletlenül hibás karaktert viszünk be, azt jelezze a program!
A feladatot próbáljuk úgy megoldani, hogy nem fordítunk a következő oldalra!

```python
ut = input("Kérem a robot parancsait: ")
hossz = len(ut)
e,d,k,n =0,0,0,0
for i in range(hossz):
    if ut[i] == 'E':
        e += 1
    elif ut[i] == 'D':
        d += 1
    elif ut[i] == 'K':
        k += 1
    elif ut[i] == 'N':
        n += 1
    else:
        print('Nem megfelelő kód!')
y=(e-d)
x=(k-n)
```

```python
if x < 0:
    for i in range(abs(x)):
        print('N',end='')
else:
    for i in range(x):
        print('K',end='')
if y < 0:
    for i in range(abs(y)):
        print('D',end='')
else:
    for i in range(y):
        print('E',end='')
```

(23d.py)

Az utolsó példa egy emelt szintű érettségi feladat megoldása. Először a leírást kell alaposan átolvasni, és megérteni!

A feladat címe: Építményadó

Egy Balaton-parti önkormányzat építményadót vezet be. Az adó mértéke a telken lévő építmény alapterületétől és a teleknek a Balatontól mért távolságától függ.

A telkeket a Balatonparttól mért távolságtól függően három sávba sorolták be. Az A sávba azok a telkek kerültek, amelyek 300 méternél közelebb vannak a tóhoz a B sáv az előzőn túl 600 méter távolságig terjed, a többi telek a C sávba tartozik. Az építmény után négyzetméterenként fizetendő összeg sávonként eltérő, azonban, ha az így kiszámított összeg nem éri el a 10.000 Ft-ot, akkor az adott építmény után nem kell adót fizetni.

```
Run:  23d
E:\00_MM\12_Python_programozas\programok\ven
2. feladat. A mintában 543 telek szerepel.
3. feladat. Egy tulajdonos adószáma: 79906
Aradi 3
A sávba 165 telek esik, az adó 20805600 Ft.
B sávba 144 telek esik, az adó 13107000 Ft.
C sávba 234 telek esik, az adó 3479600 Ft.
6. feladat. A több sávba sorolt utcák:
Besztercei
Gyurgyalag
Icce
Kurta
Rezeda
Szepesi
```

A testületi döntést az Adó Ügyosztály egy mintával készítette elő, amely csupán néhány utca adatait tartalmazza. Ezek az adatok az utca.txt fájlban vannak. A fájl első sorában a három adósávhoz tartozó négyzetméterenként fizetendő összeg található A, B, C sorrendben, egy-egy szóközzel elválasztva:

800 600 100

…

33366 Aradi 8A C 180

22510 Aradi 8B C 137

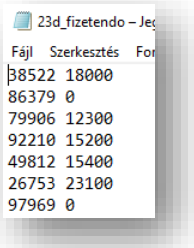
90561 Aradi 10 C 168

…

A többi sorban egy-egy építmény adatai szerepelnek egy-egy szóközzel elválasztva. Az első a telek tulajdonosának ötjegyű adószáma; egy tulajdonosnak több telke is lehet. A második adat az utca neve, amely nem tartalmazhat szóközt. A harmadik adat a házszám, majd az adósáv megnevezése, végül az építmény alapterülete következik. A minta harmadik sorában például azt látjuk, hogy a 33366 adószámú tulajdonos telke az Aradi utca 8A-ban található, és a C sávba eső telken álló építmény alapterülete 180 m2.

A fájl legfeljebb 1000 telek adatait tartalmazza. A feladat megoldása során kihasználhatja, hogy a fájlban az adatok utca, azon belül pedig házszám szerinti sorrendben következnek.
Készítsen programot, amely az utca.txt állomány adatait felhasználva az alábbi kérdésekre válaszol! A program forráskódját mentse epitmenyado.py néven! (A program megírásakor a felhasználó által megadott adatok helyességét, érvényességét nem kell ellenőriznie, és feltételezheti, hogy a rendelkezésre álló adatok a leírtaknak megfelelnek.)
A képernyőre írást igénylő részfeladatok esetén – a mintához tartalmában hasonlóan – írja ki a képernyőre a feladat sorszámát (például: 3. feladat), és utaljon a kiírt tartalomra is!
Ha a felhasználótól kér be adatot, jelenítse meg a képernyőn, hogy milyen értéket vár! Mindkét esetben az ékezetmentes kiírás is elfogadott.

1. Olvassa be és tárolja el az utca.txt állományban talált adatokat, és annak felhasználásával oldja meg a következő feladatokat!
2. Hány telek adatai találhatók az állományban? Az eredményt írassa ki a mintának megfelelően a képernyőre!
3. Kérje be egy tulajdonos adószámát, és írassa ki a mintához hasonlóan, hogy melyik utcában, milyen házszám alatt van építménye! Ha a megadott azonosító nem szerepel az adatállományban, akkor írassa ki a „Nem szerepel az adatállományban.” hibaüzenetet!
4. Készítsen függvényt ado néven, amely meghatározza egy adott építmény után fizetendő adót! A függvény paraméterlistájában szerepeljen az adósáv és az alapterület, visszaadott értéke pedig legyen a fizetendő adó! A következő feladatokban ezt a függvényt is felhasználhatja.
5. Határozza meg, hogy hány építmény esik az egyes adósávokba, és mennyi az adó összege adósávonként! Az eredményt a mintának megfelelően írassa ki a képernyőre!
6. Bár az utcák többé-kevésbé párhuzamosak a tó partjával, az egyes porták távolsága a parttól az utcában nem feltétlenül ugyanannyi. Emiatt néhány utcában – az ottani tulajdonosok felháborodására – egyes telkek eltérő sávba esnek. Listázza ki a képernyőre, hogy melyek azok az utcák, ahol a telkek sávokba sorolását emiatt felül kell vizsgálni! Feltételezheti, hogy minden utcában van legalább két telek.
7. Határozza meg a fizetendő adót tulajdonosonként! A tulajdonos adószámát és a fizetendő összeget írassa ki a mintának megfelelően a fizetendo.txt állományba! A fájlban minden tulajdonos adatai új sorban szerepeljenek, a tulajdonos adószámát egy szóközzel elválasztva kövesse az általa fizetendő adó teljes összege.



23d.py

```python
from pprint import pprint

epitmenyek = []
with open('./nyersanyag/23d_utca.txt', 'r', encoding='utf-8') as fajl:
    adok = fajl.readline().strip().split()
    adosavok = {'A': int(adok[0]), 'B': int(adok[1]), 'C': int(adok[2])}
    for sor in fajl:
        adatok = sor.strip().split()
        epitmeny = {'adoszam': adatok[0],
                    'utca': adatok[1],
                    'hazszam': adatok[2],
                    'adosav': adatok[3],
                    'terulet': int(adatok[4])}
        epitmenyek.append(epitmeny)
# print(adosavok)
# print(epitmenyek)

# 2. feladat
print(f'2. feladat. A mintában {len(epitmenyek)} telek szerepel.')

```

23d.py

```python
# 3. feladat
adoszam = input('3. feladat. Egy tulajdonos adószáma: ')
talalat = False
for epitmeny in epitmenyek:
    if adoszam == epitmeny['adoszam']:
        print(epitmeny['utca'], epitmeny['hazszam'])
        talalat = True
if not talalat:
    print('Nem szerepel az adatállományban')


# 4. feladat
def ado(ado_savonkent, adosav, alapterulet):
    ado = alapterulet * ado_savonkent[adosav]
    if ado < 10000:
        return 0
    else:
        return ado


# 5. feladat
ado_osszesites = {'A': [0, 0], 'B': [0, 0], 'C': [0, 0]}
for epitmeny in epitmenyek:
    ado_osszesites[epitmeny['adosav']][0] += 1
    ado_osszeg = ado(adosavok, epitmeny['adosav'], epitmeny['terulet'])
    ado_osszesites[epitmeny['adosav']][1] += ado_osszeg
# print(ado_osszesites)
print(f'A sávba {ado_osszesites["A"][0]} telek esik, az adó {ado_osszesites["A"][1]} Ft.')
print(f'B sávba {ado_osszesites["B"][0]} telek esik, az adó {ado_osszesites["B"][1]} Ft.')
print(f'C sávba {ado_osszesites["C"][0]} telek esik, az adó {ado_osszesites["C"][1]} Ft.')

# 6. feladat
utca_sav = {}
for epitmeny in epitmenyek:
    if epitmeny['utca'] in utca_sav:
        utca_sav[epitmeny['utca']].add(epitmeny['adosav'])
    else:
        utca_sav[epitmeny['utca']] = set(epitmeny['adosav'])
# print(utca_sav)
print('6. feladat. A több sávba sorolt utcák:')
for utca in utca_sav:
    if len(utca_sav[utca]) > 1:
        print(utca)

# 7. feladat
ado_tulajdonosonkent = {}
for epitmeny in epitmenyek:
    if epitmeny['adoszam'] in ado_tulajdonosonkent:
        ado_tulajdonosonkent[epitmeny['adoszam']] += ado(adosavok, epitmeny['adosav'], epitmeny['terulet'])
    else:
        ado_tulajdonosonkent[epitmeny['adoszam']] = ado(adosavok, epitmeny['adosav'], epitmeny['terulet'])
# pprint(ado_tulajdonosonkent)
with open('./nyersanyag/23d_fizetendo.txt', 'w', encoding='utf-8') as fizetendo:
    for azonosito in ado_tulajdonosonkent:
        print(azonosito, ado_tulajdonosonkent[azonosito], file=fizetendo)
```