Utoljára Frissítve: **2019.09.30** *dőlt szöveg*

1 - Python Language (1. videó)

Interpretált, értelmező nyelv, a CPython nevű interpretert használjuk. Léteznek más interpreterek vagy akár fordítók is, de arról majd később. "Dynamically typed", a fájlokat felülről lefelé értelmezi. Van egy `__main__()` nevű függvény, ami konvenció szerint a belépési pontja a programoknak.

Indendation - azaz tabulálás

A legtöbb nyelv ; (pontossveszzőt) használ vagy { } (kapcsos zárójelt), hogy utasításokat, logikai egységeket, függyvényeket válasszon el egymástól. Pythonban a tabulálás a szintaktika része, nem opcionális! A pontosvessző helyett legtöbbször új sort, vagy nagyobb logikai egységnél 2 újsort használunk az elválasztáshoz.

C, C++, C#, Javascript

```
if (x>y)
  {
    do something;
  }
```

Python

```
If x < y :
  do something
```

4 szóköz vagy tab, üres sorok, és PEP-8 jellemzi a szintaktikát. Ha pl. Nodepad++-ban vagy szimpla text editorban fejlesztünk, akkor nagyon kell rá figyelni. Szerencsére a VS.code és a python extension ezt már észrevétlenül kezeli.

Pythonban minden objektum. A szám típusoktól, a listaszerű tipusokon át, minden.

2 - Numbers (2., 3., 4. videó)

Numbers: int, float, complex - mindegyik egy osztály, és amit létre hozunk objektumot, az annak egy példánya.

+, -, /, * -szokásos // -div, % -mod

```
a = 8
b = 3
print(8%3)
```

 # Numbers are Immutable

Az "id" az egy egyedi belső azonosító, hasonlóan mint egy memóriacím, de nem igazi memóriacím, nem lehet rajta keresztül változózóra hivatkozni, mintha egy pointer lenne.
https://docs.python.org/3.5/library/stdtypes.html#numeric-types-int-float-complex

Pythonban az Int is egy osztály, így nincs "egyszerű típus", az Int nincs a regiszterek méretéhez kötve mint a "C" típusú nyelvekben (32, 64bit stb.), csak a memóriánk mérete szab határt, így túlcsordulás sincs. Minden a memóriában foglalódik le (a heapen és nem a stacken) - összehasonlítva C++-al pl. nem egy mutatót hozunk létre és annak tipusának megfelelő helyet foglalunk le, hanem egyből az egész memória terület lesz maga az objektum, típustól függetlenül.

```
# Int
number = 12
print(f" Integer: {number}, memory address: {id(number)}")

print(type(number))
```

```
# Float
number = 16.1
print(f" Float: {number}, memory address: {id(number)}")

print(type(number))
```

```
# Complex
number = 3 + 4j
print(f" Complex: {number}, memory address: {id(number)}")

print(type(number))
```

3 - Stringek (5. - 11. videó)

https://docs.python.org/3.5/library/stdtypes.html#string-methods

Slicing - "darabolás?" A stringeket mint egy tömböt is felfoghatjuk, ez a technika érvényes az (indexelhető) lista-szerű objektumokra is, lásd később.

```
 szoveg = D  a  r  a  b  o  l  á  s  !
          0  1  2  3  4  5  6  7  8  9
```

```
         -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
```

```
szoveg = "Darabolás!"

# szöveg[mettől:meddig:lépésköz]
# visszafelé -1-től ameddig csak tart, -1-es lépésközzel
print(szoveg[-1::-1])

print(szoveg[-1:-4:-1])

print(szoveg[1:6:2])

print(szoveg[1:])
```

```
     arabolás!
```

```
print(szoveg[2:5])

print(szoveg[2])
```

```
     r
```

```
print(szoveg[-8])
```

Népszerű string függvények és példák (11. videó)

```
szoveg = "indul a görög aludni"

print(szoveg.replace(" ", ""))

print(szoveg[::-1])

# A 8. pozíciótól indul, ezt kapjuk vissza eredményként
print(szoveg.find("görög"))

#2 db ö betű van benne
print(szoveg.count("ö"))

#nagybetűs
print(szoveg.upper())

#kisbetűs
print(szoveg.lower())

#split listába
lista = szoveg.split()
```

```
print(lista)
```

Ezzel kilistázhatjuk egy objektumon értelmezett összes attributumot vagy függvényt.

```
print(dir(int))
```

Ha egy specifikus osztályhoz tartozó leírásra vagyunk kíváncsiak, akkor ezt lekérdezhetjük, még google sem kell hozzá :)

```
print(help(str.count))
```

Stringek immutable? (8., 9., 10. videó)

```
szoveg = "alma"
print(szoveg)
print(f"szoveg id: {id(szoveg)}")

szoveg = "alma2"
print(szoveg)
print(f"szoveg id: {id(szoveg)}")
```

A stringek is "immutable"-ök azaz, ha megváltoztatom az értékét, akkor egy másik memóriahelyre íródik. Ezzel szemben a lista mutable, azaz megváltoztatható. A gyakorlatban ez annyit tesz, hogy míg a

```
szoveg = szoveg.replace(" ", "")
```

-nél figyelni kell arra, hogy a módosítás miatt egy új string objektumot kapunk vissza, ezért azt el kell "tárolni", fel kell címkézni egy változóval.

A Listánál pedig a módosítás eredménye, nem egy új objektumban, hanem magában, ugyan abban a listában fog érvényesülni:

```
lista.reverse()
```

```
szoveg = "indul a görög aludni"
szoveg = szoveg.replace(" ", "")
mirror = szoveg[::-1]

print(szoveg)

if szoveg == mirror:
    print("palindrom")
else:
```

```
    print("nem jó!")

lista = [1, 2, 3, 4, 5]
print(lista)

lista.reverse()

print(lista)
```

4 - Logikai műveletek (12., 13. videó)

```
egyenlő:         ==
nem egyenlő:     !=
kisebb:          <
nagyobb:         >
kisebb egyenlő:  <=
nagyobb egyenlő: >=

Mindig igazzal tér vissza: True
Mindig hamissal tér vissza: False

Logikai operátorok:

ÉS: and
VAGY: or
Negálás: not

print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

if else, elif van. Nincs switch case!

Két objektumot a következőképpen hasonlíthatunk össze: a dupla egyenlőségjellel, vagy az "is" kifejezéssel.

== - érték szerint hasonlítja össze a két objektumot

is - azonos ojektumról beszélünk-e, azonos-e az id-je, így nyilván érték szerint azonos lesz.

A szerkesztéshez kattintson duplán (vagy használja az Enter billentyűt)

```
print(type(True))
```

```
szoveg = "Hello2"
masikszoveg = "Hello2"



print(szoveg==masikszoveg)


print(szoveg is masikszoveg)


print(id(szoveg))
print(id(masikszoveg))
```

if és If else

```
elem = 1

if elem == 1:
    print("valami")
```

```
elem = 1

if elem == 1:
    print("1")
else:
    print("Nem talált!")
```

Inline, vagy "egy soros" IF

```
elem = 2

# alapból azt mondom, hogy az üzenet legyen = "1" ha a feltétel igaz akkor nem változik me
uzenet = "1" if elem == 1 else "Nem talált!"
print(uzenet)
```

```
Nem talált!
```

Pythonban nincs switch case, de az élet nem áll meg: "Switch case" helyett "elif".

```
elem = 3

if elem == 1:
    print("1")
elif elem == 2:
    print("2")
elif elem == 3:
```

```
    print("3")
else:
    print("Nem talált!")
```

5 - Ciklusok - for, while (17. videó)

```
for x in range(2):
  print(x)
```

```
0
1
```

```
for x in range(1, 20, 2):
  print(x)
```

```
lista = ["alma", "körte", "barack"]

for _ in lista:
  print(_)
```

```
alma
körte
barack
```

```
for x in range(10):
  print(x)
  if x == 11:
    break
else:
  print("Ha a végéig futott a ciklus, akkor ez az ág lefut")
```

```
x = 0
while x < 10:
  print(x)
  x += 1
```

```
import random

#7 db (mert range(1,8) az 7) véletlen szám generálása, 5 és 11 közötti egész értékekkel, a
lista_szamok = [random.randrange(5, 11) for x in range(1, 8)]
print(lista_szamok)

#nincs ismétlés
lista_nincs_ismetles = random.sample(range(10), 10)
print(lista_nincs_ismetles)
```

```
[6, 8, 10, 7, 7, 5, 10]
[2, 8, 7, 0, 9, 4, 1, 5, 3, 6]
```

```
a, b, c, d = 8, 3, "Hello", [1, 2, 4, 5]

print(type(c))
```

```
<class 'str'>
```

```
szam = pow(2, 128)
print(szam)
```

6 - Összetett típusok (List, Tuple, Dict, Set) (14. - 16., 18. - 23. videó)

https://docs.python.org/3/tutorial/datastructures.html

List [] - Tömbszerű láncolt lista

Tuple () - immutable List, ugyan az mint a List, csak nem lehet megváltoztatni az értékét

Dict {} - key, value pair, hasonlóan mint más nyelvekben

Set {} - "unordered collection of unique element", azaz mint a matematikai set, minden elem csak egyszer szerepelhet, a sorrend nem számít, azaz nincs sorrend.

LIST

A legjobb jellemzése: Tömbszerű láncolt lista. Az implementációja láncolt lista (saját magának foglal memóriát), de lehet rá pozícióra is hivatkozni :) Mindeféle típust tartalmazhat. (pl. egyszerűt, öszetettet - akár listában listát is, fügvényt, modult, osztályokat) https://docs.python.org/3.5/tutorial/datastructures.html#more-on-lists A "slicing" technika itt is működik.

```
lista = [1, 2, 3, 4, 5]
print(id(lista))
```

```
print(lista[::-1])
```

```
print(lista[3])
```

```
lista[3]=22
```

```
print(id(lista))

lista = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]


# az alábbi sort engedve, leméri a végrehajtási idejét ennek a blokknak
# %%timeit

uj_lista = []

for elem in lista:
  if elem > 30:
    uj_lista.append(elem)


print(uj_lista)
```

List comprehension, a python LINQ-ja, vagy SQL szerű lekérdezése. gyorsabb lehet mint a "hagyományos"

```
# az alábbi sort engedve, leméri a végrehajtási idejét ennek a blokknak
# %%timeit

l = [
    elem #select
    for elem in lista #from
    if elem>30 #feltétel
]

print(l)
```

Mutable vs Immutable

A lista 3. eleme megváltozott a Lista id-je nem változott, mert tényleg ugyan azt az objektumot módosítottuk, nem úgy mint pl. a szám típusoknál, ahol minden egyes értékadásnál új id-t kapunk. A Mutable az mutálható :) azaz megváltoztatható, az immutable pedig nem. Vagy másképp: módosítható, de minden egyes alkalommal egy új memóriacímre létrehoz egy másik objektumot. Ez néha költséges, néha szükséges, néha jó, ezek nyelvi sajátosságok. Van egy olyan irány, hogy csak immutable objektumokkal dolgozzunk, mert az sok mindent leegyszerűsít, legalábbis a párhozamos szálak kezelését biztosan. Vannak persze helyzetek, amikor nagyon költséges lenne minden egyes alkalommal egy új objektumot létrehozni, gondoljunk csak egy játék karakterére, ahol másodpercenként akár 120-szor is megváltoztatjuk az egyik attributumát. Mint más nyelvekben is, a saját típusoknál (osztályoknál) mi kontrolálhatjuk, hogy az módosítható legyen pl. csak "getter"-t állítunk be egy attribútumra, de settert már nem.

https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747

https://softwareengineering.stackexchange.com/questions/151733/if-immutable-objects-are-good-why-do-people-keep-creating-mutable-objects

```
lista = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
print(lista)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
lista.append('szoveg')  # hozzáad egy elemet, ami bármilyen típus lehet
print(lista)
```

```
utolso_elem = lista.pop()  # visszaadja és kiveszi a lista utolsó elemét (mint ha verem le
# pop(index) - nem csak az utolsó elemet lehet kihalászni, akkor az már nyilván nem verem
print(utolso_elem) # kírja az utolsó elem tartalmát, ami ugye a "szöveg"
print(lista)       # lista-t kiírja, immáron "szöveg" nélkül
```

Ha az indexet is ki szeretnénk íratni, használjuk a beépített enumerate függvényt. Figyeljük meg, hogy ez 2 értékkel tér vissza, az indexel és magával az elemmel. A "for" után ezért két változó nevet adunk meg. (ez egyébként az unpacking)

```
for index, elem in enumerate(lista):
  print(f"{index} {elem}")
```

```
0 10
1 20
2 30
3 40
4 50
5 60
6 70
7 80
8 90
9 100
```

https://docs.python.org/3/tutorial/datastructures.html

List [] - Tömbszerű láncolt lista

Tuple () - immutable list

Dict {} - key, value pair, hasonlóan mint más nyelvekben

Set {} - "unordered collection of unique element", azaz mint a matematikai set, minden elem csak egyszer szerepelhet, a sorrend nem számít

```
tuple_pelda = (1, 2, 3)
list_pelda = [1, 2, 3]
dict_pelda = {"a" : 1, "b" : 2, "c" : 3}
set_pelda = {1, 2, 3}
ures_set = set()
```

Tuple unpacking

```
# list
nevek = ["Peter", "Andrea", "David", "Marcell"]


# tuple
nevek = ("Peter", "Andrea", "David", "Marcell")


# ?tuple
nevek = "Peter", "Andrea", "David", "Bálint"


print(type(nevek))
print(nevek)
```

Fentebb már láttuk, ezt a fajta értékadást, amikor a bal oldalon is több változó van, ezt hívják unpacking-nek is.

```
nev1, nev2, nev3, nev4 =  nevek
print(nev1, nev2, nev3, nev4)
print(nev1)


#compare tuples

nevek = "Peter", "Andrea",["körte", "alma"], "David", "Marcell"
nevek2 = "Peter", "Andrea",["körte", "alma"], "David", "Marcell"

print(type(nevek2))
print(type(nevek))

if nevek==nevek2:
  print("egyforma")


egyelemu = (1,)

print(egyelemu)
print(type(egyelemu))

     (1,)
     <class 'tuple'>


#tuple, vagy két lista összehasonlítása, for ciklussal "hagyományosan"

egyforma = True

if len(nevek) != len(nevek2):
  egyforma = False
else:
```

```
  for i in range(len(nevek)):
    if nevek[i] != nevek2[i]:
        egyforma = False
        break

print(egyforma)
```

Zip és összetett típusok praktikus értékadása

```
# a zippel, két listát tudunk összeilleszteni, ha améret nem stimmel, nem baj, azokat kiha
szamok = "1 2 3 4 5 6 7 8 9 0".split()
szam_nevek = "egy kettő három négy öt hat hét nyolc kilenc".split()

print(szamok)
result = zip(szamok, szam_nevek)
print(dict(result))


#zip összemergel, két listát, tuple-t... méret különbözet nem számít

egyforma = True

for nevek, nevek2 in zip(nevek, nevek2):
    if nevek != nevek2:
        egyforma = False
        break

print(egyforma)
```

7 - Függvények - létrehozása, scope, számológép (24 - 34. videó)

```
def fuggveny():
  pass


def fuggveny(nev):
  print(f"Hello {nev}!")

fuggveny("Peter")


# pozíció és névszerinti paramétermegadás
def fuggveny(nev, becenev="Nincs"):
  if becenev=="Nincs":
    print(f"Hello {nev}!")
  else:
    print(f"Hello {becenev}!")
```

```
fuggveny("Peter", "Nincs")

from enum import Enum

class Muvelet(Enum):
    Összeadás = 1
    Szorzás = 4

def szorzas(szamok):
    result=1
    print(type(szamok))
    for _ in szamok:
        result *= _
    return result

def szamologep(*args, muvelet=Muvelet.Összeadás):
    if muvelet==Muvelet.Összeadás:
        print(sum(args))
    elif muvelet==Muvelet.Szorzás:
        print(szorzas(args))


szamologep(2, 3, 5, 10, 210, muvelet=Muvelet.Szorzás)


def adatok(**kwargs):
    print(kwargs)

adatok(nev = "Peter", kedvenc_turborago_automarkaja= "Tesla", eletkor = 12)


def adatok(*args, **kwargs):
    for key, value in kwargs.items():
        print(key, value)
    print(kwargs)


adatok(nev = "Peter", kedvenc_turborago_automarkaja= "Tesla", eletkor = 12)



x = "global változó"

def parameter_scope():
    global x
    x = "belső valtozó"
    print(x)

parameter_scope()
print(x)

print(sum((1, 2, 3, 4, 5)))

def sum():
    pass
```

```
kosar = ["alma", "körte", "dinnye"]
sztring = "ami ugye immutable"

def telikosar(sztring):
    sztring="uborka"
    print(sztring)

telikosar(sztring)
print(sztring)
```

8. if __ name __ == "__ main __":

```
from enum import Enum


class Muvelet(Enum):
    Összeadás = 1
    Szorzás = 2


def szorzas(szamok):
    result = 1
    for _ in szamok:
        result *= _
    return result


def szamologep(*args, **kwargs):
    print(kwargs)
    if kwargs["muvelet"] == Muvelet.Összeadás:
        print(sum(args))
    if kwargs["muvelet"] == Muvelet.Szorzás:
        print(szorzas(args))


def adatok(**kwargs):
    print(kwargs)


x = "globális változó"


def parameter_scope():
    x = "belső változó"
    print(x)

    def belso():
        x = "legbelsőbb"
        print(x)
```

```
    belso()


if __name__ == "__main__":

    parameter_scope()
    print(x)
    print(sum((1, 2, 3, 4, 5)))

    adatok(nev="Peter", kedvenc_turborago_automarkaja="Tesla", eletkor=12)
    szamologep(2, 3, 8, 9, muvelet=Muvelet.Szorzás)
```

9. import module

```
from szamologepmodul import szamologep, Muvelet
import sys

szamologep(2, 3, 8, 9, muvelet=Muvelet.Szorzás)
print(sys.path)
```

Modul importálására az "import modulnev" Ami többnyire egy másik python file