

Tartalomjegyzék

- 1 Ismétlés
- 2 Input / Output
 - ◆ 2.1 Help
 - ◆ 2.2 Output
 - ◆ 2.3 Input
 - ◆ 2.4 File kezelés
- 3 Feladatok
 - ◆ 3.1 1. Hatványok kiírása
 - ◆ 3.2 2. Telefonkönyv
 - ◆ 3.3 3. Telefonkönyv file
 - ◆ 3.4 4. Telefonkönyvből olvasás
- 4 Osztályok
 - ◆ 4.1 5. Osztály feladat - Telefonkönyv

Ismétlés

- Függvény definiálás:

```
def fv_neve(x, y) # fv definiálás
    return x + y
```

```
fv_neve(2, 10) # fv meghívás
```

- Listák:

```
l = [4, 5, 7, 8, 9, 4, 6]
print l[2:5]
```

- String, tuple, dictionary:

```
s = "The quick brown fox jumps over the lazy dog." # string
t = (4, 5) # tuple
```

```
d = {"alma":6, "korte":8, "banan":24} # dictionary
```

- Vezérlési szerkezetek:

```
x = 6
y = 43
if y > 21:
    print "y nagyobb"
elif y == 21:
    print "y egyenlo"
else:
    print "y kisebb"

l = [5, 4, 6, 2]
for i in l:
    print i ** 2
# 25 16 36 4
```

```
for i in range(len(l)):
    print l[i] ** 2
        # 25 16 36 4

i = 0
while i < 5:
    print i ** 3
        # 0 1 8 27 64
```

</pyhon>

* Futtatás:

** Interaktív: `'''python'''`

** File-ből: `'''python file.py'''` vagy ha adtuk futtatási jogot a file-nak, és a file első két sor

<python>

`#!/usr/bin/python`

`#coding=UTF-8`

Akkor a `./file.py` is működik.

Input / Output

Help

Interaktív python-ban meghívva a **help()** függvényt egy segítség menübe jutunk. Próbáljátok ide beírni a következőket:

- keywords
- if
- random

Output

Output már szerepelt a **print** kulcsszóval. Ez ugye bármilyen beépített típust képes volt kiírni, de ha formázni szeretnénk a kiírást akkor több dolog kell.

- **str()** - string-et csinál más típusú objektumból, **print változo** és **print str(változo)** ekvivalens
- **repr()** - nagyon hasonló a **str()**-hez, de kicsit többet mutat, próbáljátok ki: **print str(2.5656454534543654)** és **print repr(2.5656454534543654)**
- A stringek **format()** metódusával a C **printf** függvényéhez kicsit hasonlóan kezelhetjük a stringeket, pl:

```
print 'Volt egyszer egy {}, elment a {}'.format('kiskutya', 'vasarba')
```

- A kapcsos zárójeleket lecseréli a **format**-nak adott stringekkel.
- A kapcsos zárójelekbe írhatunk indexeket is, pl:

```
print 'Volt egyszer egy {1}, elment a {0}'.format('vasarba', 'kiskutya')
```

- Az indexek után **:-ot** téve további formázást adhatunk a string-nek, pl:

```
print '{0:15} ==> {1:15} = {2:15f}'.format('i = sqrt(-1)', 'i ** i valos', 0.207879)
```

- Így legalább 15 karakter méretet foglalnak el a kiírásban az adott részek

Input

Input még nem szerepelt, két függvény van amit meg fogunk nézni:

- **input()** - Kiírja a neki adott string-et, és vár egy bementet, a visszatérési értéke a bemenet automatikusan értelmezve, pl:

```
m = input("Magassag centimeterben:")
print "Magassag meterben:", m / 100.0
```

- A következő kód viszont hibát fog okozni:

```
n = input("Neved:")
print "Hello", n
```

- Ez azért van, mert az input azonnal értelmezi a beadott adatot, és a string-eket nem tudja felismerni, ő ekkor változónévre gondol.

- **raw_input()** - Kiírja a neki adott string-et, és vár egy bementet, a visszatérési értéke a bemenet stringként, pl:

```
n = raw_input("Neved:")
print "Hello", n
```

File kezelés

A file kezelés annyiban hasonlít a C-hez, hogy írásra vagy olvasásra kell megnyitnunk a file-okat, de sokkal hozzáférhetőbb. Függvények:

- **open(file_neve, megnyitاسai_mod)** - megnyit egy file-t, első paramétere a file neve, második a megnyitási mód: 'w', 'a', 'r' a szokásosak. Pl:

```
f = open('test.txt', 'w')
```

- **write()** - metódusa a file objektumoknak (az előző példában **f** file objektum), a neki adott string-et a file-ba írja. Pl:

```
f.write("Which witch watches which witch's watches?\nKovetkezo sor\n")
```

- **read()** - metódusa a file objektumoknak, beolvassa az egész file-t egy (potenciálisan) jó nagy string-be. Pl:

```
f = open('test.txt', 'r')
s = f.read()
print s
```

- **readline()** - metódusa a file objektumoknak, beolvas egy sort a file-ból, az újsor jelet is beolvassa, az üres sorokat '\n'-ként olvassa be. Ha a file végére ért üres sort olvas be. Pl:

```
line = f.readline()
print line
```

- **readlines()** - metódusa a file objektumoknak, beolvassa az összes sort mint listát. Pl:

```
lines = f.readlines()
for line in lines:
    print line
```

- **Megjegyzés:** a file objektumok iterálhatóak, azaz az előző példa ekvivalens ezzel:

```
for line in f:
    print line
```

- **close()** - metódusa a file objektumoknak, bezárja a file-t. Pl:

```
f.close()
```

Feladatok

1. Hatványok kiírása

Írjatok python scriptet, ami bekér egy pozitív egész számot, majd kiírja a számok négyzetét 1-től kezdve eddig az egész számig, szépen formázva. Például az 10-re ezt a kimenetet adja:

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

A sor behúzások legyenek kellően nagyok, hogy pl 100-ig szépen nézzen ki. Bónusz: hogyan lehetne megoldani, hogy bármilyen (értelmesen) nagy számra jól nézzen ki?

2. Telefonkönyv

Írjatok python scriptet, ami bekér egy nevet és egy telefonszámot, majd ezt megismétli még 4-szer. A telefonszámot reprezentálhatjuk string-el, vagy egész számmal ez rajtatok múlik. A neveket és telefonszámokat egy-egy listába tárolja el, majd kiírja szépen formázva.

3. Telefonkönyv file

Az előző scriptet írjátok át úgy, hogy a telefonszámokat egy telefonkonyv.txt-be mentse el, a formázást itt jobb ha hanyagoljátok, soronként egy név és a hozzá tartozó telefonszám legyen pl vesszővel elválasztva.

4. Telefonkönyvből olvasás

Most írjunk egy python scriptet, ami az előző scriptünk által írt file-ból beolvassa az adatokat ugyanolyan listába mint azok a 2. feladatban voltak tárolva. Használjátok a stringek split() metódusát, erre egy példa:

```
s = "Volt egyszer egy kiskutya"
s2 = s.split()
print s2          # ['Volt', 'egyszer', 'egy', 'kiskutya']

s = "Volt egyszer egy kiskutya, elment a vasarba."
s2 = s.split(',')
print s2          # ['Volt egyszer egy kiskutya', ' elment a vasarba']
```

Osztályok

Az osztályok egységbe zárnak és a felesleges információt elrejtik. Például a file objektumaink file típusúak, meg van bennük írva a **write()**, stb metódus, de el van rejtve előlünk a működése, mert nem kell tudnunk hogy van megvalósítva ahhoz hogy használni tudjuk. Osztályok változókból és metódusokból állnak.

Saját osztályt így definiálhatunk:

```
class MyClass:
    valt = 5

    def fv(self):
        return self.valt * 2
```

Itt a **valt** változó az osztályhoz tartozik, így elérhetjük például

"http://wiki.math.bme.hu"http://wiki.math.bme.hu"http://wiki.math.bme.huMyClass.valt"http://wiki.math.bme.hu"http://w

Viszont, ha a **fv**-t próbáljuk hasonlóképpen elérni hibába ütközünk. Ez azért van, mert az osztályt nem közvetlenül szokták használni, hanem példányosítják:

```
x = MyClass()          # létrehozunk egy MyClass típusú objektumot
x.fv()                 # így már menni fog a fv metódus
```

Így definiált objektumokhoz később is hozzávehetünk még adattagokat:

```
x = MyClass()
x.ujValt = "Valami"
print x.ujValt
#print MyClass.ujValt ez nem menne, mert az ujValt az x objektumhoz tartozik és nem az osztályához
```

Osztályoknak léteznek speciális metódusai, ezekről részletesebben következő gyakorlaton lesz szó, most nézzük csak az **__init__**-et. Ez az a függvény, ami meghívódik amikor létrehozunk egy ilyen típusú objektumot, azaz pl az **x = MyClass()** parancsnál ez hívódik. Aki nem hiszi próbálja lefuttatni ezt:

```
class Test:
    def __init__(self):
        print "Mostmar elhiszed?"

x = Test()
```

Mi ez a **self** amit mindig megadunk a metódusoknak? Ezzel a változóval hivatkozhatunk magára az objektumra amin meghívtuk az adott metódust, példából jobban érthető:

```
class Pelda:
    def fv(self, szam):
        self.v = szam

x = Pelda()
y = Pelda()
x.fv(5)
y.fv(15)

print x.v
print y.v

# Ez a fv metódus olyan volt, mintha már az x létrehozása után azt írtam volna, hogy x.v = 5
```

Amint látjátok a **self** változót nem kell nekünk átadnunk amikor meghívjuk a metódusokat, ez automatikusan

megtörténik. (Pelda.fv(x, 5) <=> x.fv(5))

Ha az **__init__** függvény nem csak a self-et, hanem más változókat is kap, akkor ezeket is meg kell adnunk, pl:

```
class Test:
    def __init__(self, szam):
        self.x = szam

x = Test(15)
print x.x
```

5. Osztály feladat - Telefonkönyv

Egészítsétek ki a következő osztályt, hogy egy ilyen objektum létrehozásakor megadhassunk neki két listát amikben nevek és telefonszámok szerepelnek, ezeket tárolja el adattagként. Valamint a **hozzavesz** metódusa vegyen hozzá egy nevet és a hozzá tartozó telefonszámot a listákhoz.

```
class Telefonkonyv:
    def __init__(self, nevek, telefonszamok):
        pass

    def hozzavesz(self, nev, telefonszam):
        pass
```

A **pass** kulcsszó csak azért van ott, mert pythonban minden blokkban kell lennie legalább egy parancsnak. A **pass** nem csinál semmit csak kitölti a teret.