Python

4. gyakorlat

Strohmayer Ádám

Agenda

- Függvények bővebben
- Dunderek
- Generátorfüggvények
- Dekorátorok
- Modulok

Függvények

```
>>> def function(*args, **kwargs):
...     print(args, kwargs)
```

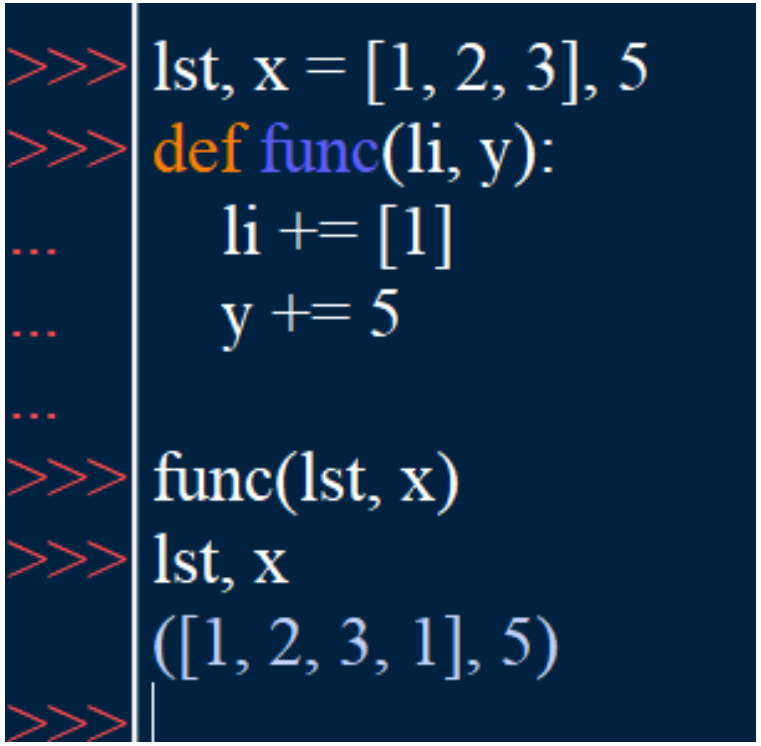
```
>>> text = "Hello World!!!"
>>> dir(text)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
_', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'ca
```

```
>>> def person_creator(age: int, name = "Reginald"):
...     return age*2, name
...
>>> person_creator("9")
('99', 'Reginald')
>>>
```

Elsőosztályú objektumok!

Paraméterátadás

- **Érték, vagy referencia szerinti?**
  - **Adattípustól függ!**
    - Mutable referencia szerint adódik át
    - Immutable lemásolódik a függvény belsejében!



- **Nem mindig mellékhatásmentesek!**

- Mutable másolásának elkerülése : lst[:]

- **Függvény túlterhelés nincs! Felül fogja írni az utoljára írt függvény az előzőt!**

Default argumentumok

- Alap értékeket ad a függvénynek!
- Nevesítés **kell!**
- **Default paraméterek előtt kell legyenek a pozícionálisak.**
- Paraméterek felcserélhetőek, ha van nevük és nem **pozícionálisak.**

```
>>> def person_creator (age = 9, name = "Reginald"):
...     return age*2, name.upper()
...
>>> person_creator()
(18, 'REGINALD')
>>> person_creator(name = "Adam", age=11*2)
(44, 'ADAM')
>>> person_creator(name="capakutyi")
(18, 'CAPAKUTYI')
>>> person_creator("yes man")
('yes manyes man', 'REGINALD')
>>>
```

*args, **kwargs

- **Pozicionális** argumentumok vannak legelől!
- Nevesítés csak **kulcs-érték argumentumoknál kell!**
- **args, kwargs bármennyi paramétert tud fogadni.**
  *args : tuple, **kwargs : dictionary formában tárolja az adatokat
- Paraméterek felcserélhetőek, ha van nevük és **nem pozícionálisak**.

```
>>> def person_creator(name, *interests, **family):
...     print(f"Name: {name}", f"Hobbies: {interests}", f"Family: {family}", sep = "\n")
...
>>> person_creator("Reggie", "reggae", "sports", dog="Amigo", cat="Kitty")
Name: Reggie
Hobbies: ('reggae', 'sports')
Family: {'dog': 'Amigo', 'cat': 'Kitty'}
>>>
```

Kérdések

Melyik függvényhívás lesz helyes?

```
>>> def person_creator(name = "Reginald", *interests, **family):
...     pass
```

person_creator()

person_creator(name = "Reggie", name = "Felix")

person_creator("sport", "singing", mental = "great")

person_creator("Adam", bunny = "Judy")

Kérdések

Melyik függvényhívás lesz helyes?

```
>>> def person_creator(name = "Reginald", *interests, **family):
...     pass
```

person_creator(interests = ["music", "sports"])

person_creator("Reggie", name = "Reginald")

person_creator(bunny = "Judy", fox = "Nick", "sports")

Mutable default paraméterek

Ne használjunk ilyet!

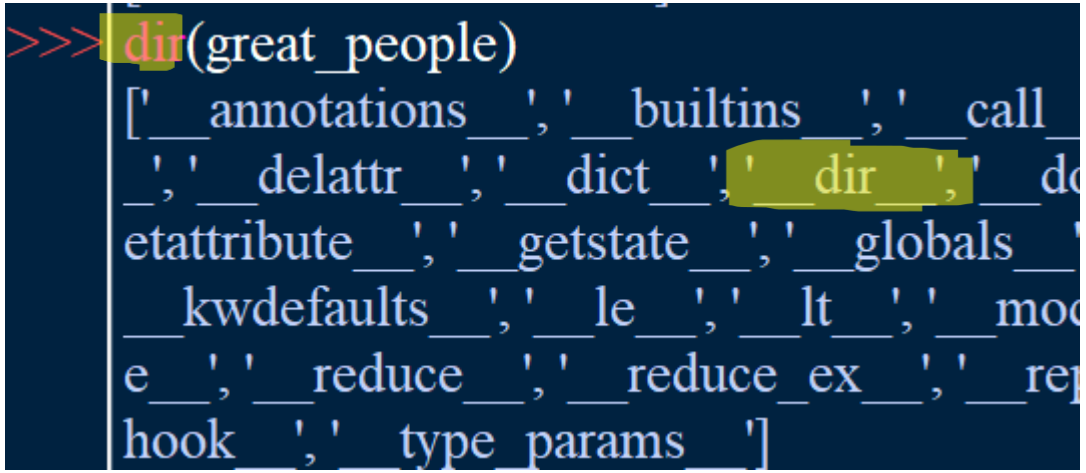
Sokszor nem várt eredményt kaphatunk!

Ha mindenképpen hasonlót akarunk, lokálisan definiáljuk az adatszerkezetünket, alapértelmezett érték None legyen!

```
>>> def great_people(person, people = []):
...     people.append(person)
...     return people
...
>>> great_people("bill")
['bill']
>>> great_people("eckhart")
['bill', 'eckhart']
>>> great_people("lembke")
['bill', 'eckhart', 'lembke']
>>>
```

Dunderek

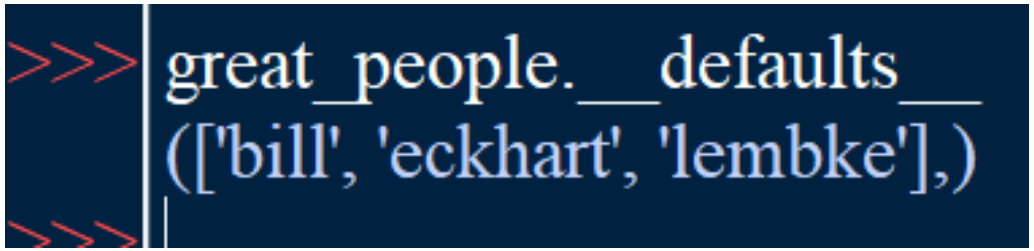
Speciális metódusok, változók!



Szintaxis:

__dunder__

Osztályok/objektumok részei – sokszor definiálni kell



Generátorfüggvények

Kulcsszava: yield

iterátort ad vissza egyszeri eredmény helyett,

mindig kiértékel!

```
>>> def normal_function(x):
...     return x**2
...
>>> type(normal_function)
<class 'function'>
```

```
>>> def gen_function():
...     yield "First call!"
...     yield "Second call!"
...     yield "I am done!"
...
>>> type(gen_function())
<class 'generator'>
>>> type(gen_function)
<class 'function'>
```

Generátorfüggvények - példa

Az utolsó yieldtől fogja folytatni a függvényt!

ha yielddel találkozik – értékkel visszatér
yieldtől folytatja újra

Ha nincs több yield – **StopIteration kivétel**

Listagenerátoros formában is létezik!
(Generator expression)

```
>>> def square_nums():
...     x = 1
...     while True:
...         yield x*x
...         x += 1
...
>>> sq_gen = square_nums()
>>> next(sq_gen)
1
>>> next(sq_gen)
4
>>> next(sq_gen)
9
>>> next(sq_gen)
16
>>>
```

```
>>> gen_lst = ( x**x for x in range(10_000))
>>> normal_lst = [ x**x for x in range(10_000)]
>>> gen_lst.__sizeof__()
184
>>> normal_lst.__sizeof__()
85160
```

```
>>> a, b, _, d, *e = (x for x in range(100))
>>> print(a, b, d, '\n', e)
0 1 3
 [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1
```

Generator expression vs. List comprehension

```
>>> import timeit
>>> timeit.timeit("""[i**i for i in range(100)]""", number=10000)
0.6330117999996219
>>> timeit.timeit("""(i**i for i in range(100))""", number=10000)
0.004520399999819347
```

Hátránya – nem indexelhető, néha nehezebb kezelni

Folyton érkező értékeket (pl. fájlolvasás, streamelés) **hatékonyan tud kezelni!**

Magasabb rendű függvények

Olyan függvény, aminek paramétere vagy visszatérési értéke (matematikai értelemben vett) függvény.

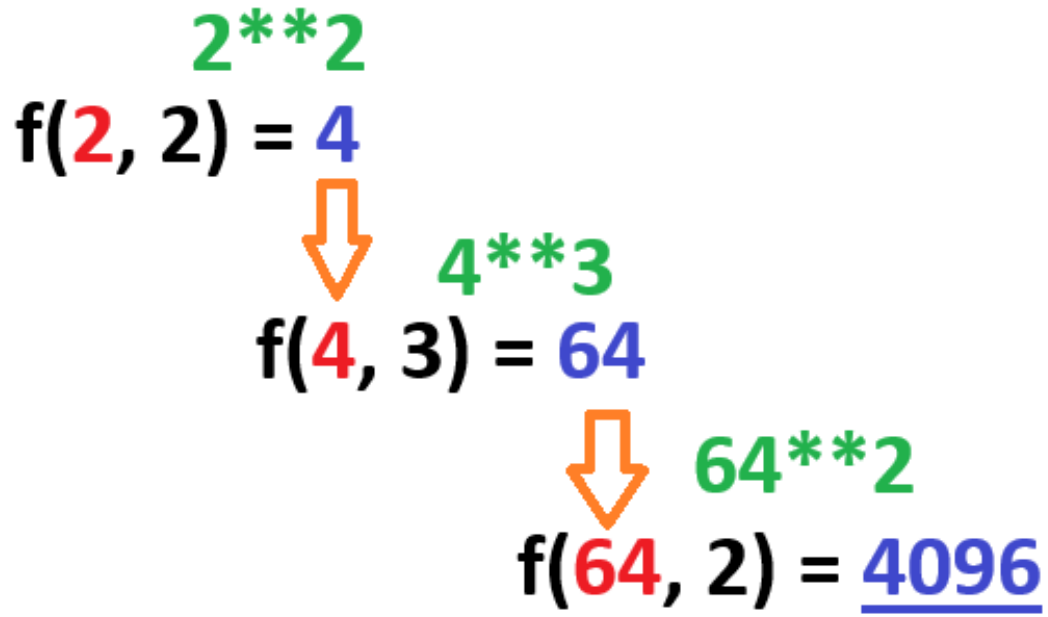
```
>>> lst = [1, 2, 3]
>>> list(map(lambda x: x**2, lst))
[1, 4, 9]
>>>
```

```
>>> lst = [x for x in range(100)]
>>> list(filter(lambda x: x > 80 and x % 2 == 0, lst))
[82, 84, 86, 88, 90, 92, 94, 96, 98]
>>>
```

```
>>> from functools import reduce
>>> gen = (x for x in range(101))
>>> reduce(lambda acc, x : acc + x, gen)
5050
>>>
```

```
>>> from functools import reduce
>>> lst = [31, 32, 21, 32, 41, 0, -1]
>>> reduce(lambda mini, x : x if x < mini else mini, lst)
-1
```

Reduce



reduce(lambda x,y : x**y, [2, 3, 2], 2)

reduce(lambda x,y : x**y, [2, 3, 2]) ???

További magasabb rendű függvények

max/min(iterable, key=függvény) – mi szerint válassza a maxot

sorted(iterable, key=függvény) – adatszerkezet rendezése függvény szerint

pl. sorted(["hellooo", "world", "fridaaay"], key=len)
hosszúság szerint lesz rendezve az adatszerkezetünk

```
>>> h = ["hellooo", "world", "fridaaay"]
>>> sorted(h, key=len)
['world', 'hellooo', 'fridaaay']
>>>
```

```
>> grades = {"Dam" : 2, "Reggie" : 5, "Jimmy" : 3}
>> sorted(grades.items(), key=(lambda kv_pair : kv_pair[1]))
[('Dam', 2), ('Jimmy', 3), ('Reggie', 5)]
```

```
>>> lst = [(1,2,3), (3,2,1)]
>>> max(lst, key = lambda x: (x[1], x[2]))
(1, 2, 3)
```

Dekorátorok

Módosítják egy függvény viselkedését!

Függvény átkerül a belső függvénybe, ott fog módosulni a viselkedése.

A paraméterszám a dekorátor belső függvényétől függ innentől!

```
>>> def evil_decorator(function):
...     def inner(param1, param2):
...         #egyeznie kell a function paraméterszámával!
...         print(f"{function.__name__} >:)")
...         result = function(param1, param2)
...         return result * 7
...     return inner
...
>>> @evil_decorator
... def adder(x, y):
...     return x + y
...
>>> adder(5, 5)
adder >:)
70
>>>
```

Modulok

Logikai egységek

Importálás kétféleképpen:

import math

függvényhíváskor hivatkozni kell a modulra!
pl. math.sin(5)

from math import *

függvényhíváskor nem kell hivatkozni a modulra!

csak adott dolgokat importál ekkor!
pl. from math import sin
sin(5) használható

Package, modul

Csomag (package) letöltése: pip install csomagnév

egy csomagban lehet több modul!

törlése: pip uninstall csomagnév

Modul/fájl importálás működése

minden importálódik – változók, osztályok, függvények is! (kivéve, ha nem publikusak)

alias – import numpy as np

np-vel lehet innentől a modul műveleteire hivatkozni!

Külön fájlok futtatása

__init__.py tárolja el packageként a fájljainkat!

Csomagként értelmeződik az adott mappánk tőle!

Futtatáskor speciális dunderek – pl. __name__

ha egy adott fájlt futtatunk – a fájl name dundere __main__ lesz!

Hasznos lehet tesztelésnél, különbséget tehet importált, közvetlenül futtatott fájlok között!

Konfliktusok importálásoknál

Ugyanolyan nevű függvények összeakadhatnak!

my_module.py > sin

```python
def sin(x):
    print(f"You have sinned, {x}!")
```

Saját fájlban függvény

module_test.py

```python
from math import sin
from my_module import sin

sin(3.14)
```

Utoljára importált lesz érvényben!

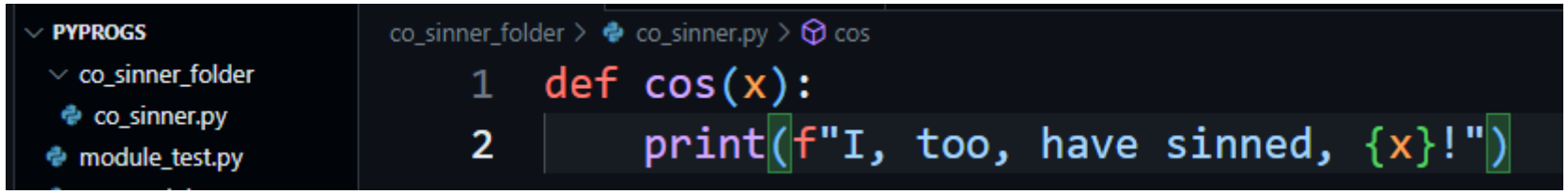
```
>python module_test.py
 You have sinned, 3.14!

>_
```

Különböző csomagverzióknál: **dependency hell**

Importálás más mappából, lokálisan

Futtatástól relatív kell megadni a modulunk helyét!




```
module_test.py
1  from math import cos
2  from co_sinner_folder.co_sinner import cos
3
4  cos(3.14)
```

```
>python module_test.py
 I, too, have sinned, 3.14!
```

Importálás csomagból, modulból

Csomag egy nagyobb logikai egység a modulnál!

Csomagból importálhatunk modulokat:

from package import module1

de külön, más **nyilvános** objektumot is!

from package.module2 import funfunction, iclass

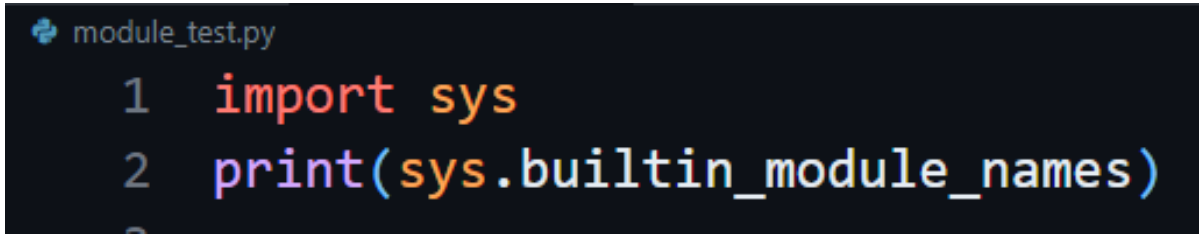
Frommal specifikusan egy névtér alá vonjuk a beimportált objektumokat!

Fontosabb modulok

sys, os – rendszerszintű műveletek

math, random – matematikai műveletek

pandas, numpy, matplotlib – adatelemzéshez



Beépített modulok lekérdezése

Mi történt eddig?

- Beszéltünk a különböző függvényargumentumokról.
- Beszéltünk a dunderekről.
- Beszéltünk generátorfüggvényekről.
- Néztünk példákat magasabb rendű függvényekre.
- Megbeszéltük mi a dekorátor.
- Beszéltünk a modulok fontosabb tulajdonságairól, működéséről.
- Hoztunk példát fontosabb modulokra.

Feladatok Canvasben!

Köszönöm a figyelmet!