funkcionális programozás

map, reduce, filter, lamda

Készítette: Vastag Atila

2020

map

A map függvény az első argumentumként megadott függvényt (*increment*) hívja meg az argumentumként megadott szekvencia vagy szekvenciák mindegyik elemére (*numbers*).

A map egy listát ad visszatérési értékként, ami az egyes számítások eredményeit tartalmazza.

```
from typing import *

def increment (x: int) -> int:
    return x + 1

numbers: Set[int] = {1 , 2 , 3, 4 , 5 , 6}

result: List[int] = map(increment, numbers)

print(list(result))

#output
[2, 3, 4, 5, 6, 7]
```

map

Ha több szekvenciát adunk meg argumentumként, akkor a függvénynek annyi paraméterrel kell rendelkezni, ahány szekvenciát argumentumként megadtunk.

```
from typing import *

def sum (x: int, y: int) -> int:
    return x + y

  parossak: Set[int] = {0, 2, 4, 6, 8}
paratlanok: Set[int] = {1, 3, 5, 7, 9}

result: List[int] = map(sum, parossak, paratlanok)

print(list(result))
```

```
#output
[1, 5, 9, 13, 17]
```

reduce

A **reduce** függvény egy szekvenciát redukál úgy, hogy annak mindegyik elemére rekurzívan alkalmaz egy függvényt. Az első paraméterként megadott függvénynek két argumentuma kell, hogy legyen. A **reduce** függvénynek lehet egy opcionális harmadik paramétere is, ami a rekurzív számolás kezdoértéke.

```
from typing import *
from functools import reduce

def sum (x: int, y: int) -> int:
    return x + y

numbers: Set[int] = {1, 2, 3, 4}

result: int = reduce(sum, numbers)
print(result)   #output 10

result = reduce(sum, numbers, -5)
print(result)   #output 5
```

A *reduce()* függvény az alábbi módon hajtódik végre:

(((1 + 2) + 3) + 4) => 10

filter

A **filter** az első argumentumként megadott függvényt alkalmazza a második argumentumként megadott szekvencia minden egyes elemére és visszatérési értékként egy olyan új listát ad, ami a szekvencia összes elemét tartalmazza, melyekre a függvény visszatérési értéke true.

```
from typing import *

def pozitivE (x: int) -> bool:
    return x > 0

numbers: Set[int] = {1, -2, -5, -9, 4}

result: List[int] = filter(pozitivE, numbers)
print(list(result))

#output
[1, 4]
```

lambda

A **lambda** egysoros névtelen függvény.

```
#vegyünk példának egy egyszerű függvényt,
#mely négyzetre emeli a megadott számot
def negyzetreEmeles(x: float) -> float:
    return x ** 2

print(negyzetreEmeles(3))

#output
9

#nézzük most ugyanezt a feladatot lambda kifejezéssel
negyzet = lambda x: x**2

print(negyzet(2))

#output
4
```

lambda

```
#vegyünk példának egy egyszerű függvényt két paraméterrel,
#mely összeadja ezt a két számot
def osszeadas(x: float, y: float) -> float:
    return x + y

print(osszeadas(3, 7))

#output
10
```

```
#nézzük most ugyanezt a feladatot lambda kifejezéssel
osszeg = lambda x, y: x + y
print(osszeg(2, 8))

#output
10
```

lambda

```
#vegyünk példának egy rendezésre
def keyFunction(x:tuple):
    return x[1]

adatok = [(20, "korte"), (30, "szilva"), (40, "alma")]
rendezett = sorted(adatok, key = keyFunction)

print(rendezett)

#output
(40, 'alma'), (20, 'korte'), (30, 'szilva')]
```

```
#nézzük most ugyanezt a feladatot lambda kifejezéssel
adatok = [(20, "korte"), (30, "szilva"), (40, "alma")]
rendezett = sorted(adatok, key = lambda x: x[1])

print(rendezett)

#output
(40, 'alma'), (20, 'korte'), (30, 'szilva')]
```