

Tartalomjegyzék

- 1 Listákról még
- 2 Listaértelmezések
- 3 Ismétlés a feladatok előtt
 - ◆ 3.1 Függvények használata függvényekben
 - ◆ 3.2 Listák manipulálása függvényekkel
 - ◆ 3.3 Szummázás adott értékig
- 4 CloudCoder feladatok

Listákról még

- Listák a listában (mátrix):

```
M = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- Ennek első eleme:

```
print M[0]    # [1, 2, 3]
```

- Tehát **M** elemei listák, így ezeknek is lekérhetjük az elemeit:

```
print M[0][2]    # 3
```

- Ugyanúgy adhatunk hozzá elemeket, mint korábban:

```
M.append([10, 11, 12])
print M    # [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

- A belső listák hossza eltérhet:

```
N = [[1, 2, 5], [4], [3, 2, 1, 4]]
```

- Bár ez ritka, de keverten szinte bármit tárolhatunk listákban:

```
H = ["kutya", [1, 5, "macska"], [3, [5, 3]], 5.3]
```

Listaértelmezések

```
[kifejezés for elem in bejárható_objektum]
```

Egy olyan listát hoz létre melyben a **kifejezés** szerepel a **bejárható_objektum** minden elemére.

```
[kifejezés for elem in bejárható_objektum if feltétel]
```

Mint az előző, de csak azok az elemek lesznek benne melyekre teljesül a **feltétel**.

```
[kifejezés for elem1 in bejárható_objektum1 if feltétel1
 for elem2 in bejárható_objektum2 if feltétel2
 for elemN in bejárható_objektumN if feltételN]
```

Több feltétel és ciklus is írható akár.

Pl:

```
[x ** 2 for x in [1, 2, 3, 4]] # [1, 4, 9, 16]
[x for x in [-1, 2, -3, 4] if x > 0] # [2, 4]
```

Ismétlés a feladatok előtt

Függvények használata függvényekben

Használhatunk minden általunk definiált függvényt a függvényinkben. Pl:

```
def hatvany(x, n):
    return x ** n

def fv(x):
    return hatvany(x, 3) + 2 * hatvany(x, 2) + 2 * x + 1
```

Collatz sejtés feladat clean code stílusban:

```
def paros(n):
    if n % 2 == 0:
        return True
    else:
        return False

def collatz_lepes(n):
    if paros(n):
        return n / 2
    else:
        return n * 3 + 1

def collatz(n):
    while n != 1:
        n = collatz_lepes(n)
    print n
```

Listák manipulálása függvényekkel

- Tekintsük a következő függvényt, mely egy számokat tartalmazó listát kap és vissza szeretné adni a listát melyben a számok négyzetei vannak:

```
def negyzet1(L):
    for e in L:
        e = e ** 2
    return L

L = [1, 3, 5]
print negyzet1(L) # [1, 3, 5]
```

Nem úgy működik ahogy szeretnénk, mert a for ciklus **e** változója csak egy másolata a lista adott elemének, nem az elem maga, így ha négyzetre emeljük attól még a listában nem történik változás.

- Próbáljuk akkor közvetlenül elérni az elemeket indexekkel:

```
def negyzet2(L):
    i = 0
```

```
while i < len(L):
    L[i] = L[i] ** 2
    i += 1
return L
```

```
L = [1, 3, 5]
print negyzet2(L) # [1, 9, 25]
print L # [1, 9, 25]
```

Ez már négyzetre emeli az értékeket, de van egy nem kívánt mellékhatása, a bemeneti **L** listát is megváltoztatta, ezt legtöbbször nem szeretnénk.

- A jó megoldás:

```
def negyzet3(L):
    retval = []
    for e in L:
        retval.append(e ** 2)
    return retval
```

```
L = [1, 3, 5]
print negyzet3(L) # [1, 9, 25]
print L # [1, 3, 5]
```

Egy új listát töltünk fel a kívánt elemekkel és ezt adjuk vissza, így a bemeneti listát biztosan nem bántottuk.

Szummázás adott értékig

- A következő egy olyan függvény, ami szummázza a bemeneti **L** lista elemeit egészen addig amíg a szumma értéke kisebb mint 20, majd visszaadja az így kapott szummát:

```
def resz_sum(L):
    i = 0
    s = 0
    while s + L[i] < 20:
        s += L[i]
        i += 1
    return s

lista = [5, 2, 7, 8, 4, 7]
print resz_sum(lista) # 14
```

Egy másik megoldás:

```
def resz_sum2(L):
    s = 0
    for e in L:
        if s + e < 20:
            s += e
        else:
            break # megszakítja a ciklust
    return s
```

CloudCoder feladatok

- <https://ccweb.math.bme.hu/cloudcoder/>