



# Szkript nyelvek

## A Python programozási nyelv

Szathmáry László

Debreceni Egyetem  
Informatikai Kar

### 4. Gyakorlat

- mi lesz False-ként kiértékelve; sztringbuffer
- a tuple adattípus
- list comprehension
- vezérlési szerkezetek
- függvények

(utolsó módosítás: 2019. szept. 6.)

2019-2020, 1. félév



# Mi lesz hamisnak (False) értékelve?

False

None

0 # illetve 0.0

"" # vagy: ""

[]

()

{}

üres szekvenciák

üres szótár

```
>>> bool(None)
False
>>> bool([])
False
>>> bool([1,2,3])
True
>>> bool("py")
True
```

Minden egyéb érték igaznak (True) számít.

HF: XOR.

<http://docs.python.org/3/library/stdtypes.html#truth-value-testing>

# Sztringbuffer

**Példa:** vegyük az egész számokat 1-től 15-ig s írjuk le őket egymás mellé.  
Az eredményt sztringként kezeljük: „123456789101112131415”.

# Sztringbuffer

**Példa:** vegyük az egész számokat 1-től 15-ig s írjuk le őket egymás mellé.  
Az eredményt sztringként kezeljük: „123456789101112131415”.

Naiv megközelítés:

```
5 res = ""
6 for i in range(1, 15+1):
7     res += str(i)
8 #
9 print(res)
```

Sztringbuffer alkalmazása:

```
5 parts = []
6 for i in range(1, 15+1):
7     parts.append(str(i))
8 #
9 res = "".join(parts)
10 print(res)
```

# tuple

```
4 >>> a = (1, 2, 3)
5 >>> a[0]
6 1
7 >>> a[0] = 5
8 Traceback (most recent call last):
9   File "<stdin>", line 1, in <module>
10  TypeError: 'tuple' object does not support item assignment
11 >>>
12 >>> m = ('Total Recall', 1990, 7.5)
13 >>> m
14 ('Total Recall', 1990, 7.5)
15 >>> len(m)
16 3
17 >>> m[:2]
18 ('Total Recall', 1990)
19 >>>
20 >>> (x, y) = (1, 2)
21 >>> x
22 1
23 >>> y
24 2
25 >>> single = ('hi',)
26 >>> single
27 ('hi',)
```

immutable  
(read-only)

párhuzamos értékadás  
(a zárójel elhagyható itt)

1-elemű tuple  
(figyeljünk a szintaxisra)

HF: [tuple02.py](#) kiegészítése.

```

4  def get_movie_info():
5      # contact database,
6      # fetch info from a website, etc.
7      return ('Total Recall', 1990, 7.5)
8
9
10 def main():
11     (title, year, score) = get_movie_info()
12     print('Title:', title)
13     print('Year:', year)
14     print('Score:', score)

```

ezzel a módszerrel egy  
függvény **több** értéket is  
visszaadhat

value unpacking

```

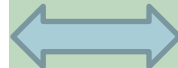
1  >>> a = 3
2  >>> b = 9
3  >>> a
4  3
5  >>> b
6  9
7  >>> a, b = b, a
8  >>> a
9  9
10 >>> b
11 3
12 >>>

```

Két változó értékének  
a felcserélése.  
Nincs szükség segédváltozóra :)

## List comprehension: listák előállításának egy kompakt módja

```
1 >>> nums = [1, 2, 3, 4]
2 >>> squares = [n*n for n in nums]
3 >>> squares
4 [1, 4, 9, 16]
```



```
1 >>> nums = [1, 2, 3, 4]
2 >>> squares = []
3 >>> for n in nums:
4 ...     squares.append(n*n)
5 ...
6 >>> squares
7 [1, 4, 9, 16]
```

Általánosan:

```
[expr for var in list]
```

Opcionális „if”:

```
1 >>> nums = [8, 3, 2, 1, 5, 9, 2]
2 >>> small = [n for n in nums if n <= 2]
3 >>> small
4 [2, 1, 2]
5 >>>
```

csak azokat tartja meg,  
melyekre teljesül a  
feltétel

# Feladat

A következő feladatokat „list comprehension” segítségével kellene megoldani.

...

Link: <https://arato.inf.unideb.hu/szathmary.laszlo/pmwiki/index.php?n=Py3.20120818d>



## Gyakori művelet: szekvencia megfordítása

```
1 >>> s = 'Python rulz'
2 >>> s[::-1]
3 'zlr nohtyP'
4 >>>
5 >>> li = [23, 76, 84, 12]
6 >>> li[::-1]
7 [12, 84, 76, 23]
8 >>>
```

a végétől az elejéig egyesével  
lépkedünk visszafele  
(új szekvenciát ad vissza)

## Hatványozás; int és long típusok

```
1 >>> 2**3
2 8
3 >>> 2**64
4 18446744073709551616L
5 >>> 2**128
6 340282366920938463463374607431768211456L
7 >>>
8 >>> import sys
9 >>> sys.maxint
10 9223372036854775807
11 >>> a = sys.maxint + 1
12 >>> a
13 9223372036854775808L
14 >>> type(sys.maxint)
15 <type 'int'>
16 >>> type(a)
17 <type 'long'>
```



tetszőlegesen nagy egész  
számok kezelhetők Pythonban  
(csak a memóriától függ)

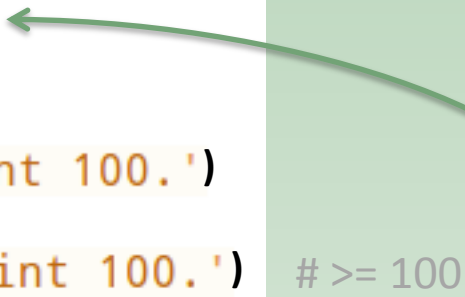
itt: 64 bites int ( $2^{63} - 1$ )

túlcsordulás helyett  
automatikusan *long* típusra  
vált

**Python 3:** csak *int* típus van, de ez  
nagy számok esetén *long*-ként  
viselkedik.

## if / elif / else

```
4 def main(num):  
5     if num < 0:  
6         print('Negativ szam.')  
7     elif num == 0:  
8         print('Nulla.')  
9     elif num < 100:  
10        print('Kisebb mint 100.')  
11    else:  
12        print('Nagyobb mint 100.') # >= 100
```



„else if” helyett „elif”-et írunk. Ezt úgy tudjuk könnyen megjegyezni, hogy az „elif” ugyanolyan hosszú, mint az „else” szó.

# for

```
4 def main():  
5     li = ['alfa', 'beta', 'gamma']  
6  
7     for e in li:  
8         print(e)  
9  
10    for index, e in enumerate(li):  
11        print(index, e)
```

alfa  
beta  
gamma

0 alfa  
1 beta  
2 gamma

Gyakori eset, hogy az elemek indexére is szükségünk van. Ekkor használjuk az `enumerate()` fv.-t, ami Python 3-ban szintén egy iterátor.

Lásd még `enumerate(lista, start=1)`.

# break / continue pass

```
4  def main():
5      cnt = 0
6      while True:
7          cnt += 1
8          if cnt == 42:
9              break
10     #
11     print(cnt)    # 42
12
13     li = ['ananasz', 'banan', 'citrom']
14     for e in li:
15         if e == 'banan':
16             continue
17         #
18         print(e)
19
20
21  def palindrom():
22      pass    # TODO...
```

42

ananasz  
citrom

üres utasítás

The diagram illustrates the execution flow of the provided Python code. Green arrows point from comments on the right to specific lines of code: one arrow points from '42' to line 11, another from 'ananasz' and 'citrom' to line 18, and a third from 'üres utasítás' to line 22.

# docstring

A dokumentáció lehet egy (azaz 1) soros:

```
4 def square(num):  
5     """Return the square of a given number."""  
6     return num ** 2
```

Vagy több soros. Ekkor az első sor legyen egy rövid összefoglaló. Utána hagyjunk ki egy üres sort, majd részletezzük a függvény célját, működését, mellékhatásait, stb.

```
9 def square_v2(num):  
10     """Return the square of a given number.  
11  
12     Calculate the square of the input number."""  
13     return num * num
```

(lásd még: F függelék)

Szokjunk meg a programjaink dokumentálását!

## opcionális paraméterek / alapértelmezett paraméter értékek

```
>>> def greet(name, greetings="Hello"):
...     print("{g}, {n}!".format(g=greetings, n=name))
...
>>> greet("Laszlo")
Hello, Laszlo!
>>>
>>> greet("Laszlo", greetings="Hola")
Hola, Laszlo!
>>>
>>> greet("Laszlo", "Bonjour")
Bonjour, Laszlo!
```

## opcionális paraméterek (további példák)

```
4 def hello(name, repeat=1, postfix=''):
5     for i in range(repeat):
6         print(name + postfix)
7
8
9 def main():
10     hello('Laci')
11     print('#' * 10)
12     hello('Laci', repeat=3)
13     print('#' * 10)
14     hello('Laci', postfix='!')
15     print('#' * 10)
16     hello('Laci', repeat=3, postfix='!')
17     print('#' * 10)
18     print('#' * 10)
19     hello('Laci', 3)
20     print('#' * 10)
21     hello('Laci', 3, '')
22     print('#' * 10)
23     hello('Laci', '') # HIBA!
```



# Feladatok

1. [[20120818a](#)] két pont közti távolság (tuple)
2. [[20120818d](#)] *list comprehensions* (befejezni)
3. [[20120910a](#)] hangrend
4. [[20120818e](#)] ezernél kisebb pozitív egész számok (PE #1)  
(ezúttal *list comprehension*-nel)
5. [[20120920e](#)] sortörés
6. [[20130211b](#)] diamond
7. [[20120818f](#)] négyzetek összege, összeg négyzete (PE #6)
8. [[20130305a](#)] XOR (Vigyázat! Gondolkodós feladat!)