

01_PythonBev_adat_alapmuv

August 22, 2024

1 Adattípusok és alpműveletek, Változók megadása

Ismerkedés a jupyter notebook felülettel. Első python programunk. Adattípusok és alpműveletek: egész, valós, (komplex), sztingek, logikai, semmi (None); és műveleteik. Változók megadása, használata.

1.1 Bevezetés

1.1.1 Programozási alapfogalmak

- **Algoritmus:** Valamely feladat megoldására alkalmas véges hosszú lépéssorozat.
 - A fogalom hétköznapi feladatokra is alkalmazható (pl. Sacher-torta készítés, könyvespolc takarítás :-).
- **Adatszerkezet:** Adatelemek tárolására és hatékony használatára szolgáló séma (példa: lista).
- **Programozási nyelv:** Szigorú szabályokra épülő nyelv, melynek segítségével az ember képes a számítógép felé kommunikálni az utasításait.
- **Programozás:** Algoritmusok és adatszerkezetek megtervezése illetve megvalósításuk valamilyen programozási nyelven (kódolás).

1.1.2 A Python nyelv jellemzői

+ szintaxisa tömör, elegáns + könnyen tanulható (“brain-friendly”) + több 10 ezer külső csomag érhető el hozzá (<https://pypi.org/>) + erős közösség, évente PyCon konferenciák + szabadon használható, nyílt forráskódú + platformfüggetlen + értelmezett nyelv, típusai dinamikusak + többparadigmás nyelv – bizonyos feladatokhoz lassú lehet – többszálú lehetőségei korlátozottak

1.1.3 Történelem

- **1994:** A Python 1.0 megjelenése.
- **2000:** A Python 2.0 megjelenése.
- **2001:** A Python Software Foundation megalakulása.
- **2003:** Az első PyCon konferencia.
- **2008:** A Python 3.0 megjelenése. Nem volt kompatibilis a 2-es verzióval. Az áttérés lassan ment, de végül megtörtént.
- **2018:** Guido van Rossum lemond a BDFL címről. Egy ötfős bizottság lesz a legfőbb döntéshozó szerv a nyelvvel kapcsolatban (lásd: PEP 8016).

1.2 A Jupyter Notebook környezet (Colab)

- A **Jupyter Notebook** egy böngésző alapú, interaktív munkakörnyezet.
- Elsődlegesen a Python nyelvhez fejlesztették ki, de más programozási nyelvekkel is használható.
- Egy notebook cellákból áll, a cellák lehetnek szöveges (Markdown) vagy kód típusúak.
- A kódcellákat le lehet futtatni, akár többször is egymás után. A futtatás eredménye megjelenik az adott kódcella utáni kimenetben.
- A notebook használata kétféle üzemmódban történik:
 - Parancsmódban tudjuk elvégezni a cellaszintű műveleteket (pl. új cella beszúrása, cella törlése, cellák mozgatása, lépegetés a cellák között, stb). Néhány billentyűparancs:
 - * **b**: Új kódcella beszúrása az aktuális cella után. (**a** ugyan ez elé)
 - * **m** (**ctrl+MM**) : Az aktuális cella típusának átállítása szövegesre.
 - * **dd** (**ctrl+MD**): Az aktuális cella törlése.
 - * **Enter**: Átlépés szerkesztőmódba (az aktuális cella tartalmának szerkesztése).
 - Szerkesztőmódban tudjuk szerkeszteni a cellák tartalmát. Néhány billentyűparancs:
 - * **Shift+Enter**: Az aktuális cella futtatása.
 - * **Esc**: Visszalépés parancsmódba.
- A billentyűparancsokról a Help / Keyboard Shortcuts menü ad részletesebb leírást. (Eszközök / Használható parancsok vagy **ctrl +MH**)

1.3 Technikai részletek

1.3.1 Implementációk

- CPython (<http://python.org/>)
- PyPy (<http://pypy.org/>)
- IronPython (<http://ironpython.net/>)
- Jython (<http://www.jython.org/>)
- MicroPython (<https://micropython.org/>)

1.3.2 Telepítés

Windows

A legcélszerűbb egy Python disztribúciót telepíteni: - Anaconda (<https://www.anaconda.com/products/distribution>) - Miniconda (<http://conda.pydata.org/miniconda.html>)

Linux

Több életképes alternatíva is van: - A rendszer csomagkezelőjének használata. - Az értelmező telepítése csomagkezelővel (vagy akár forráskódból), a külső csomagok telepítése pip-pel. - Python disztribúció használata.

1.3.3 Fejlesztőkörnyezetek

nehézsúlyú - PyCharm (<http://www.jetbrains.com/pycharm/>) - Visual Studio Code (<https://code.visualstudio.com/>) - PyScripter (<https://sourceforge.net/projects/pyscripter/>) - Spyder (<https://www.spyder-ide.org/>) ...

könnyűsúlyú - Emacs / Vim / Geany / ... - IDLE (az alap Python csomag része) - Jupyter Notebook
...

1.4 Egyszerű adattípusok

1.4.1 Egész szám (int)

- A számok között a szokásos módon végezhetünk műveleteket. (+ - * /)
- [A python dokumentációban a számokra vonatkozó rész](#)
- Eredmény kiírása `print()` függvény segítségével

```
[11]: # Próbálja ki a 4 alapl műveletet!  
print(2 + 4)  
print(2 - 4)  
print(2 * 5)  
print(2 / 5)
```

6
-2
10
0.4

Megjegyzések: - A szóközök nem számítanak, a fenti írásmód a PEP 8 kódolási stílust követi. - **A sorkezdő behúzásnak viszont jelentése van a Pythonban!** - A Jupyter a futtatás után megjeleníti a cella utolsó kifejezését. - Vesszővel elválasztva az egész sor egy gyűjtemény => Mindet kiírja

Ügyeljünk a precedenciára! Azaz a műveletek szokásos sorrendjére!

A sorrend zárójelezéssel () felülbírálnak.

Feladat: Számítsa ki az alábbi műveletek eredményét!

$$(a) \frac{1}{3 + 4 \cdot 2}; \quad (b) \frac{2}{3 + 4} \cdot 10 + 5$$

```
[12]: # (a) használjunk zárójelezést ha szükséges!  
print(1 / (3 + 4 * 2))
```

0.09090909090909091

```
[13]: # (b) használjunk zárójelezést ha szükséges!  
print(2 / (3 + 4) * 10 + 5)
```

7.857142857142857

Egész értékű változó

```
[14]: # Hozunk létre egy i nevű változót, és tegyük bele a 11 értéket!  
i = 11
```

Megjegyzések: - Az = az értékadás műveleti jele. - i változó felveszi a megadott értéket, de magának az értékadásnak nincs eredménye. - Emiatt a cella kimenete üres.

```
[15]: # irassuk ki a képernyőre i értékét -- print()
      print(i)
```

11

```
[16]: # A változóra a továbbiakban is lehet hivatkozni.
      print(i + 3)
      print(3 * i)
```

14

33

```
[17]: # A változó értéke természetesen változtatható.
      i = 12
```

Az értékadást lehet kombinálni a többi művelettel.

Rövid jelölés: +=, -=, *=

```
[1]: # += *= -= használata
     i = 10
     print(i)
     i *= 2      # i = i * 2
     print(i)
     i += 3      # i = i + 3
     print(i)
```

10

20

23

Megjegyzések a műveletekről Sok hibalehetőséget megelőz, hogy külön műveleti jele van a lebegőpontos (/) és az egészosztásnak (//).

```
[19]: # Lebegőpontos osztás.
      3 / 7
```

[19]: 0.42857142857142855

```
[20]: # Egészosztás (levágja a törtrészt).
      3 // 7
```

[20]: 0

További hasznos műveletek

```
[21]: # Maradékképzés.
```

[21]: 3

```
[22]: # Van hatványozás is, ** a műveleti jele.  
2**10
```

```
[22]: 1024
```

```
[23]: # Az abszolútértéket az abs() függvény kiszámítja
      abs(-4)
```

[23] : 4

Égész számok tárolódása a memóriában **Feladat:** Mekkora a 8 bájt = 64 biten ábrázolható legnagyobb egész szám?

Pythonban egyébként nincs ilyen határ. A Python képes tetszőleges hosszúságú egész számokkal dolgozni. Mutassuk is ezt meg egy példán!

```
[24]: # 8 bájtban = 64 biten ábrázolható legnagyobb (előjeles) egész szám
max_int = 2**64 - 1
print(max_int)
```

18446744073709551615

```
[25]: # nincsen túlcsordulási hiba
max int+1
```

[25] : 18446744073709551616

```
[26]: # A Python képes tetszőleges hosszúságú egész számokkal dolgozni  
      # nincsen túlcsordulási hiba.  
      111111111111111111111111111111111111111111111111111 + 1
```

[illegible]

1.4.2 Lebegőpontos szám (float)

- A [lebegőpontos számbázis](#) lehetővé teszi a valós számokkal történő, közelítő számolást.
- A Python lebegőpontos típusa az IEEE-754 szabvány dupla pontosságú (64 bites double) típusát valósítja meg.
- [A python dokumentációban a számokra vonatkozó rész](#)

[27]: *# Lebegőpontos állandókat a tizedespont használatával tudunk megadni.*
1.25

[27]: 1.25

```
[28]: # Gyök kettő (közelítő) kiszámítása.  
2**0.5 , 2**(1/2)
```

[28]: (1.4142135623730951, 1.4142135623730951)

```
[29]: # Hozzunk létre egy f nevű, lebegőpontos típusú változót!  
f = 1.25  
f
```

[29]: 1.25

```
[30]: # A type() függvénnyel tudjuk lekérdezni f típusát.  
type(f)
```

[30]: float

```
[31]: # ...vagy bármely más érték típusát.  
type(i)
```

[31]: int

```
[32]: # Tegyük most f-be egy int típusú értéket!  
# Pythonban ez minden probléma nélkül megtehető.  
f = 100
```

```
[33]: # nézzük meg f típusát ismét!  
type(f)
```

[33]: int

1.4.3 Komplex szám (complex)

- A Python támogatja a komplex számokkal való számolást, külső könyvtárak használata nélkül.
- A j-jelöli a képzetes egységet ha szám kerül elé
- [A python dokumentációban a számokra vonatkozó rész](#)

```
[34]: # Osztás algebrai alakban.  
1/(2 + 3j)
```

[34]: (0.15384615384615385-0.23076923076923078j)

```
[35]: # A képzetes egység hatványozása.  
1j**2
```

[35]: (-1+0j)

1.4.4 Logikai érték (bool)

- A logikai igaz értéket a `True`, a hamisat a `False` jelöli.
- A nagy kezdőbetű fontos, a Python különbözőnek tekinti a kis- és nagybetűket.
- [A python dokumentációban a logikai értékre vonatkozó rész](#)

```
[36]: # Hozzunk létre logikai típusú változót!  
# Nézzük meg a típusát is!  
b = False  
type(b)
```

[36]: bool

Logikai műveletek: ÉS (`and`), VAGY (`or`), TAGADÁS (`not`)

```
[37]: # Logikai ÉS művelet. -- and  
print(True and True)  
print(True and False)  
print(False and False)
```

True
False
False

```
[38]: # Logikai VAGY művelet. -- or  
print(True or False)  
print(True or True)  
print(False or False)
```

True
True
False

```
[39]: # Logikai tagadás. -- not  
not True
```

[39]: False

Az összehasonlító műveletek (`>`, `>=`, `==`, `!=`) eredménye logikai érték.

```
[40]: # Nagyobb-e -3 mint 2?  
-3 > 2
```

[40]: False

```
[41]: # nagyobb egyenlő-e?  
3 >= 3
```

[41]: True

```
[42]: # kisebb egyenlő-e?  
3 <= 3
```

[42]: True

```
[43]: # Pythonban az egyenlőségvizsgálat műveleti jele ==.  
3 == 3
```

[43]: True

```
[44]: # != (nem egyenlő) operátor.  
3 != 3
```

[44]: False

Feladat: Definiáljunk egy `n` egész változót és vizsgáljuk meg, hogy 50-nél nagyobb, páros számot adtunk-e meg?

```
[45]: # Megoldás:  
n = 48  
(n > 50) and (n % 2 == 0)
```

[45]: False

1.4.5 None / Semmi, üres (NoneType)

- A szó jelentése *semmi* vagy *egyik sem*. A Pythonban a `None` értéknek helykitöltő szerepe van. Ezzel jelölhetjük pl. a hiányzó vagy érvénytelen eredményt vagy az alapértelmezett beállítást.
- [None a python dokumentációban](#)

```
[46]: # A None érték típusa.  
type(None)
```

[46]: NoneType

```
[47]: # Ha a cella utolsó kifejezése None értékű, akkor nincs kimenet.  
2 + 3  
None
```

1.4.6 Sztring (str)

- A sztring adattípus szöveges értékek tárolására szolgál.

- Pythonban a sztring nem más mint [Unicode](#) szimbólumok (másnéven Unicode karakterek) nem módosítható sorozata.
- (<https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str>)

```
[48]: # A sztringállandót ' jelekkel határoljuk.
s = 'alma és körte'
s
```

```
[48]: 'alma és körte'
```

- Síma idézőjel: 'allows embedded "double" quotes'
- Dupla idézőjel: "allows embedded 'single' quotes" "it's"
- Tripla idézőjel (több soros szöveg): '''Three single quotes''', """Three double quotes"""

A határoló idézőjel ' nem a sztring része, csak az adattípust jelzi!

```
[49]: # ...de lehet használni " jeleket is.
s2 = "ez egy másik szöveg"
s2
```

```
[49]: 'ez egy másik szöveg'
```

```
[50]: # Írjuk ki a sztring tartalmát, határoló jelek nélkül! print()
s3 = ' ide jön egy idézet "ez egy idézet"'
print(s3)
```

```
ide jön egy idézet "ez egy idézet"
```

```
[51]: # több soros sztring tripla idézőjellel
'''ez így egy szöveg
és itt folytatódik'''
```

```
[51]: 'ez így egy szöveg\nés itt folytatódik'
```

```
[52]: # A type() függvény most is működik.
type(s2)
```

```
[52]: str
```

```
[53]: # A sztringben természetesen használhatunk Unicode szimbólumokat.
'I  '
```

```
[53]: 'I  '
```

```
[54]: # Az újsornak és a tabulátornak is van karaktere: \n és \t
s = "alma és\n körte:\tbarack"
s
```

```
[54]: 'alma és\n körte:\tbarack'
```

```
[55]: # képernyőre kiírva látszik a hatásuk
print(s)
```

```
alma és
körte: barack
```

Milyen hosszú egy sztring? Azaz hány karakterből áll!

- Használjuk a `len()` függvényt.
- A sztring hossza a használt Unicode szimbólumok száma
- A szóköz is karakternek számít!

```
[56]: ## Hány karakterből áll az alábbi s sztring?
s = "alma"
len(s)
```

```
[56]: 4
```

Sztring karaktereinek “kinyerése” (indexelés)

- Az indexelés **0-tól indul!!!**
- `[]` között a sorszám, például: `s[0]`
- Negatív index is értelmes (végéről).

```
[57]: # s karaktereinek kinyerése. Az indexelés 0-tól indul!
s[0],s[1]
```

```
[57]: ('a', 'l')
```

```
[58]: # A kinyert karaktert egy 1 hosszú sztring formájában kapjuk vissza.
len(s) , len(s[0])
```

```
[58]: (4, 1)
```

```
[59]: # Túlindexelés esetén hibaüzenetet kapunk.
# próbáljuk ki!
s[8]
```

```
-----
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_13459/3792544162.py in <module>
      1 # Túlindexelés esetén hibaüzenetet kapunk.
      2 # próbáljuk ki!
----> 3 s[8]
```

```
IndexError: string index out of range
```

```
[60]: # negatív index is értelmes  
s[-1]
```

```
[60]: 'a'
```

```
[61]: # A sztring karaktereit nem lehet módosítani!  
s[1] = "b"
```

```
-----  
TypeError                                Traceback (most recent call last)  
/tmp/ipykernel_13459/2779301116.py in <module>  
      1 # A sztring karaktereit nem lehet módosítani!  
----> 2 s[1] = "b"  
  
TypeError: 'str' object does not support item assignment
```

```
[62]: # Természetesen s-nek adhatunk új értéket.  
s = "körte "
```

```
[63]: # Írjuk ki s tartalmát!  
print(s)
```

körte

Néhány hasznos sztring eljárás `strip()`, `lower()`, `upper()`

```
[64]: ## Szöveg fehér karakterekkel  
text = "\t Ez egy próba szöveg. \t\n"  
print(text)
```

Ez egy próba szöveg.

```
[65]: # Fehér karakterek (szóköz, tabulátor, sortörés) eltávolítása  
# a sztring elejéről és végéről.  
text.strip()
```

```
[65]: 'Ez egy próba szöveg.'
```

```
[66]: # Megadott karakterek eltávolítása a sztring elejéről és végéről.  
text.strip("\n")
```

```
[66]: '\t Ez egy próba szöveg. \t'
```

```
[67]: # Kisbetűssé alakítás.  
text.strip().lower()
```

```
[67]: 'ez egy próba szöveg.'
```

```
[68]: # Nagybetűssé alakítás.  
text.strip().upper()
```

```
[68]: 'EZ EGY PRÓBA SZÖVEG.'
```

Műveletek sztringekkel +, *, in

```
[69]: text_a = "alma"  
text_b = "körte"
```

```
[70]: # Sztringek összefűzése  
text_a + " " + text_b
```

```
[70]: 'alma körte'
```

```
[71]: # Sztring ismétlése a megadott számú alkalommal.  
'alma ' * 10
```

```
[71]: 'alma alma alma alma alma alma alma alma alma '
```

```
[72]: # Tartalmazásvizsgálat:  
print('a' in 'alma')  
print('b' in 'alma')
```

True

False

```
[73]: # Üres sztring létrehozása. Nulla karakterből áll.  
ures = ''  
len(ures), type(ures)
```

```
[73]: (0, str)
```

Sztring kódolása Hogyan kerül az Unicode karaktersorozat egy fájlba?

```
[74]: ## Sztringből a kódolás műveletével képezhetünk bájtsort.  
s = "körte "  
s_cod = s.encode("utf-8")  
print(s)  
print(s_cod)
```

```
körte
b'k\xc3\xb6rte \xe2\x99\xa5 \xe2\x99\xac'
```

```
[75]: ## Az eredmény típusa?
      type(s_cod)
```

```
[75]: bytes
```

```
[76]: ## A bájtok száma nagyobb lehet, mint a Unicode szimbólumok száma!
      len(s_cod), len(s)
```

```
[76]: (14, 9)
```

```
[77]: ## Bájt sorozatból a dekódolás műveletével képezhetünk sztringet.
      s_cod.decode("utf-8")
```

```
[77]: 'körte  '
```

Feladat:

- Hány bájton tárolódnak a magyar ábécé ékezetes kisbetűi UTF-8 kódolás esetén? Próbáljunk ki néhány esetet!
- Hány bájton tárolódik a és a szimbólum?

```
[78]: print(len("é".encode("utf-8")))
      print(len(" ".encode("utf-8")))
```

```
2
3
```