SZKRIPTNYELVEK – PYTHON SEGÉDLET

Ez a jegyzet az első Szkriptnyelvek zárthelyi dolgozatra való felkészüléshez hivatott segítséget nyújtani. A segédletben összefoglalom a **Python** nyelv fontosabb elemeit, kódpéldákon szemléltetve azokat.

NOTE: A jegyzet NEM helyettesíti sem a gyakorlaton, sem az előadáson elhangzottakat!

1. Python kód írása és értelmezése

A forráskódot tetszőleges szerkesztőprogramba írjuk, és `.py` kiterjesztéssel mentjük el.

Ha a programot terminálból (parancssorból) futtatjuk, akkor azt a `python program_neve.py` utasítással tesszük meg, ahol `program_neve` az elmentett fájlnak a neve.

Mivel a gyakorlatokon Python3-mal foglalkozunk, viszont az Irinyi Kabinetben az alapértelmezett Python értelmező a Python2, ezért a ZH alatt használjuk a **`python3 program_neve.py`** utasítást fordításkor!

2. Alapvető nyelvi elemek

A. Kommentek

- `# ez egy egysoros komment`
- ```
  ''' ez egy több
      soros komment '''
  ```

B. Kiíratás

`print(…)` : kiírja a paraméterül kapott kifejezést

- tetszőleges számú paraméterrel hívható (vesszővel elválasztva)
- alapértelmezett módon sort tör a kiíratás végén – ha ezt nem szeretnénk, akkor utolsó paraméterként adjuk át neki az `end=""` utasítást (pl. `print("Hello", end="")`)

```
print("Hello World!")
print("Never", "gonna", "give", "you", "up")
```

```
Hello World!
Never gonna give you up
```

C. Fontosabb adattípusok

- `bool`: logikai adattípus (`True` vagy `False` értéke lehet)
- `int`: egész szám
- `float`: lebegőpontos (valós) szám
- `str`: string
  - megadás: aposztrófok (`'…'`) vagy idézőjelek (`"…"`) között
  - összefűzés: + operátorral

```
print('malteros ' + "lapát")  # összefűzés
```

```
malteros lapát
```
```

D. Változók

- Létrehozás, értékadás: `valtozonev = ertek`
  - a változónév érzékeny a kis- és nagybetűkre
  - a változónév csak betűvel vagy alulvonással (_) kezdődhet
- Változó értékének kiíratása: `print(valtozonev)`
- **Dinamikusan típusosság:** ugyanaz a változó eltérő típusú értékeket is tárolhat

```
val = 5          # val típusa: int
print(val)

val = "Béla"     # val típusa: string
print(val)
```

```
5
Béla
```

E. Fontosabb operátorok

- **Aritmetikai operátorok:** `+` (összeadás), `-` (kivonás) `*` (szorzás), `/` (valós osztás), `//` (egész osztás), `%` (maradékos osztás), `**` (hatványozás)
- **Hozzárendelő operátorok:** =, +=, -=, *=, /= (ugyanúgy működnek, mint C-ben)
- **Összehasonlító operátorok:** == (egyenlő), != (nem egyenlő), < (kisebb), <= (kisebb vagy egyenlő), > (nagyobb), >= (nagyobb vagy egyenlő)
- **Logikai operátorok:** `and` (és), `or` (vagy), `not` (nem)

NOTE: Pythonban a ++ és -- operátorok **nem** szerepelnek! Helyettük használjuk a `+= 1`, illetve `-= 1` utasításokat az érték 1-gyel történő növeléséhez / csökkentéséhez

F. Beolvasás

`input(prompt)`: beolvasás az alapértelmezett bemenetről

- `prompt` értékeként megadható egy string, ami az input bekérésekor fog megjelenni
- alapértelmezett módon stringet olvas be → szükség esetén castolni kell
  - `int(s)`: az `s` stringet int értékké alakítja
  - `float(s)`: az `s` stringet float értékké alakítja
- stringet csak stringgel lehet összefűzni! → más adattípusokat ilyenkor castolni kell
  - `str(a)`: az `a` változót stringgé alakítja

```
a = input("Első szám: ")          # beolvasás
b = input("Második szám: ")

a = int(a)            # castolás egész számra
b = int(b)

sum = a + b

print("Az összeg: " + str(sum)) # castolás stringre
```

```
Első szám: 5
Második szám: 2

Az összeg: 7
```

3. Vezérlési szerkezetek

A. Szelekciós vezérlés

- **if**, **elif** (`else if` helyett), **else** utasítások
- Pythonban **nem** használatosak a más nyelvekből ismerős, `{` és `}` által meghatározott blokkok
  - helyettük az **indentálás** dönti el, hogy mely utasítások mihez tartoznak
  - feltételek megadása: *ld. összehasonlító és logikai operátorok (2. oldal)*

```
num = input('Adj meg egy számot: ')
num = int(num)

if num < 0:                 # "ha" ág
     print("Negatív")
     a = abs(num)
     print("Abszolútértéke: " + str(a))
elif num == 0:              # "egyébként ha" ág
     print("Nulla")
else:                       # "egyébként" ág
     print("Pozitív")
```

```
Adj meg egy számot: -5
Negatív
Abszolútértéke: 5
```

B. Ismétléses vezérlés (Ciklusok)

- **while**-ciklus

```
i = 1

while i <= 10:
     print(i, end=" ")
     i += 1
```

```
1 2 3 4 5 6 7 8 9 10
```

- **for**-ciklus
  - "hagyományos" `for`-ciklus: **range**`(start, end, step)`
    - kiírja a számokat `start`-tól `(end-1)`-ig `step` lépésközzel

```
for i in range(1, 7, 2):
     print(i)
```

```
1
3
5
```

  - stringek karaktereinek bejárása

```
for char in "foo":
     print(char)
```

```
f
o
o
```

  - lista elemeinek bejárása

```
for element in [12, 23, 34]:
     print(element)
```

```
12
23
34
```

4. Függvények

- függvénydefiníció:

```
def fuggveny_neve(param1, param2, …):
    # függvény törzse
```

- függvényhívás:

```
fuggveny_neve(param1, param2, …)
```

- **return**: visszatérési érték
- a paramétereknek adhatók default értékek is

```
def osszead(a = 0, b = 0):
    return a + b
```

- **Pythonban nincs function overloading!** (még eltérő paraméterezés esetén sem!)
  - névütközés esetén a kódban legkésőbb szereplő függvény fog hívódni

```
def hello():
    print("Hello!")

def hello(name):
    print("Hello, " + name + "!")

# hello()              # error!
hello("Józsi")
```

```
Hello Józsi!
```

5. Bővebben a stringekről

- Megadásuk: aposztrófok ('…') vagy idézőjelek ("…") között
- Összefűzés: + operátorral
- Karakterek indexelése: `string_neve[index]`
  - negatív index esetén a string végéről kezd el számolni
  - az indexelés megadható intervallumosan is: `mettől:meddig:lépésköz`
    - ha nem adjuk meg, a 0. karaktertől megy az utolsóig 1-es lépésközzel
  - Pythonban a stringek **immutable**-ök (nem módosíthatók)

```
s1 = "almás"
s2 = "pite"

s = s1 + " " + s2      # összefűzés

# s[0] = "o"           # hiba! (immutable)
print(s)
print(s[0])            # 0. karakter
print(s[0:5])          # 0-4. karakter
print(s[-1])           # utolsó karakter
print(s[::])           # a string elejétől a végéig
print(s[::-1])         # string megfordítása(!)
```

```
almás pite
a
almás
e
almás pite
etip sámla
```

- `len(s)`: visszaadja az `s` string hosszát (karaktereinek számát)
- Rengeteg beépített stringkezelő függvény
  - `s.lower()`: csupa kisbetűssé alakítja az `s` stringet
  - `s.upper()`: csupa nagybetűssé alakítja az `s` stringet
  - `s.startswith(v)`: visszaadja, hogy a `v` értékkel kezdődik-e az `s` string
  - `s.endswith(v)`: visszaadja, hogy a `v` értékre végződik-e az `s` string
  - `s.count(v)`: visszaadja, hogy a `v` érték hányszor szerepel az `s` stringben
  - `s.replace(old, new)`: lecseréli az `s` stringben az összes `old` részstringet `new`-ra
  - `s.isdigit()`: igaz, ha az `s` stringben csak számjegyek vannak
  - `s.isalpha()`: igaz, ha az `s` stringben csak betűk vannak
  - `s.isalnum()`: igaz, ha az `s` stringben csak alfanumerikus karakterek vannak
  - `s.split(delim)`: feldarabolja az `s` stringet, `delim` karakterek mentén
  - …

```
szoveg = "Szeretem a Szkriptnyelveket"

print(len(szoveg))                                  # hossz lekérése
print(szoveg.lower())                               # kisbetűsítés
print(szoveg.upper())                               # nagybetűsítés
print(szoveg.count("e"))                            # hány db 'e' van benne?
print(szoveg.replace("Szkriptnyelveket", "Prog2-t"))    # lecserélés
```

```
27
szeretem a szkriptnyelveket
SZERETEM A SZKRIPTNYELVEKET
6
Szeretem a Prog2-t
```

Fun fact: ha egy stringet többször kell kiírnunk, `for`-ciklus helyett használhatjuk ezt is:

```
s = "ha" * 5
print(s)
```

```
hahahahaha
```

6. Listák

- Gyakorlatilag dinamikus méretű tömbök
- Tetszőleges, nem feltétlenül azonos típusú adatok tárolására alkalmas
- Létrehozás (példa):

```
ures = list()          # üres listát hoz létre
kutyak = ["Ubul", "Snoopy", "Marmaduke"]
```

- Listaelem lekérdezése: `lista_neve[index]`
  - az intervallumos megadás itt is működik

```
print(kutyak[1])
```

```
Snoopy
```

- Lista hosszának lekérdezése: `len(lista_neve)`

```
print(len(kutyak))
```

```
3
```

- Rengeteg beépített függvény és művelet a listák kezelésére
  - `lista.append(e)`: beszúrja az `e` elemet a `lista` végére
  - `lista.insert(pos, e)`: beszúrja a `lista pos` pozíciójára az `e` elemet
  - `lista.remove(e)`: törli a `lista`-ból a **legelső** `e` elemet
  - `del(lista[i])`: törli a `lista` `i`. indexén lévő elemét
  - `e in lista`: visszaadja, hogy `e` szerepel-e a `lista`-ban
  - `lista.sort()`: rendezi a `lista` elemeit
  - `lista.clear()`: törli a `lista` elemeit
  - …

```
hallgatok = ["Józsi", "Béla", "Sanyi", "Béla"]

hallgatok.append("Ervin")     # hozzáfűzés a lista végéhez
print(hallgatok)
hallgatok.insert(0, "Gábor")  # hozzáfűzés a lista elejéhez
print(hallgatok)

hallgatok.remove("Béla")      # az első "Bélát" törli

if "Sanyi" in hallgatok:      # "Sanyi" benne van-e a listában
      print("Sanyi hallgató")

hallgatok.sort()        # rendezés (ábécé sorrend)
print(hallgatok)

hallgatok.clear()       # lista elemeinek törlése
```

```
['Józsi', 'Béla', 'Sanyi', 'Béla', 'Ervin']
['Gábor', 'Józsi', 'Béla', 'Sanyi', 'Béla', 'Ervin']
Sanyi hallgató
['Béla', 'Ervin', 'Gábor', 'Józsi', 'Sanyi']
```

- Listák bejárása: `for`-ciklussal

Példa: Írassuk ki egy tetszőleges szöveg szavait egyenként, lista használatával!

```
szoveg = "A Kalkulus még könnyű tárgynak számít"
szavak = szoveg.split(" ")    # szóköz mentén darabolunk

count = 1          # hanyadik szót írjuk ki éppen

for szo in szavak:             # lista bejárása
      print("A(z) " + str(count) + ". szó: " + szo)
      count += 1
```

```
A(z) 1. szó: A
A(z) 2. szó: Kalkulus
A(z) 3. szó: még
A(z) 4. szó: könnyű
A(z) 5. szó: tárgynak
A(z) 6. szó: számít
```

7. Dictionary-k

- **Kulcs-érték párok**ból álló rendezett leképezés
- Ugyanúgy működik, mint Javában a map, PHP-ban az array típus vagy JavaScriptben az asszociatív tömb
- Létrehozás:

```
ures = dict()          # üres dictionary-t hoz létre
dictionary_neve = { kulcs1: ertek1, kulcs2: ertek2, … }
```

- Elem lekérdezése:

```
# egyik módszer
dictionary_neve[kulcs]

# másik módszer
dictionary_neve.get(kulcs)
```

```
jegyek = { "ASD123": 3, "FOO999": 5, "LOL420": 3 }    # dictionary

neptun = input("Neptun kód: ")

if neptun in jegyek:    # ha a beírt érték szerepel kulcsként…
      # …érték lekérdezése
      print(neptun + " érdemjegye: " + str(jegyek[neptun]))
else:
      print("Nincs ilyen Neptun kód!")
```

```
Neptun kód: ASD123
ASD123 érdemjegye: 3
```

- Dictionary bejárása: **for**-ciklussal
  - kulcsok bejárása:

```
for kulcs in jegyek:
      ...
```

  - értékek bejárása:

```
for ertek in jegyek.values():
      ...
```

  - kulcsok és értékek együttes bejárása:

```
for kulcs, ertek in jegyek.items():
      ...
```

8. Objektumorientált programozás

A. Osztályok, objektumok létrehozása

- **Osztályok:**
  - osztályok létrehozása:

```
class OsztalyNeve(object):
      # osztály törzse
```

  - adattagokat, metódusokat (tagfüggvényeket) tartalmaz
    - elérésük: `.` operátorral
    - minden metódusnak az első paramétere a `self`, amivel magára az objektumra hivatkozhatunk (ez a Javából és C++-ból ismert `this`-nek feleltethető meg)
  - **konstruktor:** példányosításkor fut le → adattagok inicializálására használatos

```
def __init__(self, param1, param2, …):
      self.adattag1 = param1  # adattagok inicializálása a paraméterekkel
      self.adattag2 = param2
      ...
```

- **Objektumok:**
  - osztály példányosítása során hozhatók létre
  - átadjuk a paramétereket a konstruktornak (az első, `self` paramétert nem)
  - objektumok létrehozása:

```
objektum_neve = OsztalyNeve(param1, param2, …)
```

B. Objektumok kiíratása

- Ha a **`print()`** utasítással kiíratjuk az objektumot, akkor valami ehhez hasonlót kapunk:

```
<__main__.OsztalyNeve object at 0x012B08D0>
```

- Ezt lehetőségünk van "szebbé tenni" a `__str__`beépített függvény felüldefiniálásával
  - az objektum kiíratásáért felel
  - egy stringgel tér vissza
    - ezt a stringet definiáljuk felül
  - lényegében úgy működik, mint Javában a `toString()`

```
class OsztalyNeve(object):

      ...

      def __str__(self):
            return "Ez a szöveg fog megjelenni az objektum kiíratásakor"
```

C. Getterek, setterek megvalósítása

- Pythonban **nincsenek** láthatósági módosítószók
- Konvenció alapján az adattag neve előtti egyszeres alulvonás (_) jelzi azt, hogy az adattag nem publikus használatra van szánva (“private”)
  - viszont ettől még kívülről ugyanúgy elérhető lesz az adattag

```
_adattag
```

- **Getter:** “private” adattagok lekérdezésére szolgáló függvény

```
def get_adattag(self):
   return self._adattag
```

- **Setter:** “private” adattagok értékének beállítására szolgáló függvény

```
def set_adattag(self, ertek):
   self._adattag = ertek
```

- A fenti szintaxis Pythonban ritkán használatos, helyettük a **property**-ket szokás használni
  - getterek property-je:

```
@property
def adattag(self):
     return self._adattag
```

  - setterek property-je:

```
@adattag.setter
def adattag(self, ertek):
     self._adattag = ertek
```

  - ezeknek köszönhetően kívülről úgy tűnik, mintha publikus adattagokkal dolgoznánk

D. Operátor felüldefiniálás

- Pythonban lehetőségünk van bizonyos operátorok működését felüldefiniálni, ha felülírjuk a nekik megfelelő metódus működését
- Néhány fontosabb metódus, amelyekkel operátorokat definiálhatunk felül:
  - `__eq__`: egyenlőség (`obj1 == obj2` operátor hívja meg)
  - `__neq__`: nem egyenlőség (`obj1 != obj2` operátor hívja meg)
  - `__add__`: összeadás (`obj1 + obj2` operátor hívja meg)
  - …

E. Típusellenőrzés

- `isinstance(obj, type)`: visszaadja, hogy az `obj` objektum `type` típusú-e
  - `True` vagy `False` visszatérési érték
  - nem csak osztályból példányosított objektumokra! (pl. `isinstance(12, int)`)

F. Objektumorientált programozás – Példa

```
# osztály létrehozása
class Sutemeny(object):
      def __init__(self, nev, szeletek = 8):    # konstruktor
            self.nev = nev                              # adattagok inicializálása
            self._szeletek = szeletek                   # "private" adattag

      @property
      def szeletek(self):                 # getter (property-s megvalósítás)
            return self._szeletek

      @szeletek.setter
      def szeletek(self, szeletek):       # setter (property-s megvalósítás)
            if szeletek > 0:
                  self._szeletek = szeletek
            else:
                  self._szeletek = 10

      def __str__(self):                # objektum kiíratásáért felelő metódus
            return "Ez egy " + str(self.szeletek) + " szeletes " + self.nev

      def __eq__(self, masik_suti):      # == operátor felüldefiniálása
            if isinstance(masik_suti, Sutemeny):      # típusellenőrzés
                  # egy egyszerű megoldás
                  return self.__dict__ == masik_suti.__dict__

            return False

      def __add__(self, masik_suti):      # + operátor felüldefiniálása
            if isinstance(masik_suti, Sutemeny):
                  uj_szeletek = self.szeletek + masik_suti.szeletek
                  uj_suti = Sutemeny("szupersüti", uj_szeletek)
                  return uj_suti
            else:
                  print("HIBA: Nem sütemény objektumot adtál át!")
# főprogram
suti1 = Sutemeny("krémes")                # példányosítás
suti2 = Sutemeny("brownie", 16)

print(suti1.szeletek)         # getter hívása
suti2.szeletek = -12          # setter hívása

print(suti1)      # __str__ hívása
print(suti2)

suti3 = suti1 + suti2   # __add__ hívása
print(suti3)

suti4 = Sutemeny("szupersüti", 18)
print(suti3 == suti4)   # __eq__ hívása
```

```
8
Ez egy 8 szeletes krémes
Ez egy 10 szeletes brownie
Ez egy 18 szeletes szupersüti
True
```

9. Kivételkezelés

- A program futása során történhet olyan kivételes esemény, ami meggátolja a program futását
  - ekkor **kivétel** (exception) dobódik
  - pl. ha a kódban 0-val szeretnénk osztani

```
num = 5 / 0
print("angry matematikus noises")
```

```
Traceback (most recent call last):
  File "app.py", line 1, in <module>
    num = 5 / 0
ZeroDivisionError: division by zero
```

- Mi is dobhatunk ilyen kivétel objektumot
  - beépített vagy általunk írt kivétel osztálypéldány egyaránt dobható
  - átadható a kivételeknek egy szöveg is
    - ez fog kiíródni hibaüzenetként

```
raise Exception("Valamit elszúrtunk…")
```

- Ha valahol el lett dobva egy kivétel, azt el is tudjuk kapni

```
try:
      # a kód azon része, ahol kivétel dobódhat
except Exception as e:
      # Exception típusú hiba elkapása (e-ként tudunk rá hivatkozni)
finally:
      # mindig lefutó kódrész
```

  - több **except** ág esetén a legelső, a dobott kivételre illeszkedő fog lefutni

```
def osztas(a, b):
      if (isinstance(a, int) and isinstance(b, int)):
            if b == 0:
                  raise ZeroDivisionError("HIBA: 0-val való osztás") # kivétel dobása
            return a / b
      raise TypeError("HIBA: Nem egész érték")  # kivétel dobása
# főprogram
try:
      a = osztas(5, 2)
      print(a)
      b = osztas(5, 0)        # ZeroDivisionError!
      print(b)
except TypeError as te:
      pass                                  # "passzoljuk" (nem csinálunk semmit)
except ZeroDivisionError as zde:
      pass                                  # "passzoljuk" (nem csinálunk semmit)
except Exception as e:
      print("Jaj!")
finally:
      print("--- Kivételkezelés vége ---")
```

```
2.5
--- Kivételkezelés vége ---
```

10. Fájlkezelés

Ebben a jegyzetben a Pythonra jellemző, ún. kontextuskezelőkkel megvalósított fájlkezelést tárgyaljuk.

Nézzünk erre egy gyakorlati példát!

Példa: A `be.txt` állomány minden sora egy-egy egész számot tartalmaz. Olvassuk be a fájl tartalmát, majd számítsuk ki a fájlban szereplő értékek átlagát! Ezt írjuk ki egy `ki.txt` nevű állományba!

```
average = 0       # változó az átlagnak

with open('be.txt', 'r', encoding='utf-8') as fp:     # be.txt megnyitása olvasásra
      line = fp.readline()          # első sor beolvasása
      count = 0                     # hány sort olvastunk be

      while line:                   # amíg van sor...
            average += int(line)
            count += 1
            line = fp.readline()          # következő sor beolvasása

average = average / count

with open('ki.txt', 'w', encoding='utf-8') as fp:     # ki.txt megnyitása írásra
      fp.write('Az átlag: ' + str(average) + '\n')          # fájlba írás
```

be.txt tartalma:
```
7
4
1
10
2
5
8
9
6
3
```

ki.txt tartalma:
```
Az átlag: 5.5
```