

Hálózatok építése és üzemeltetése

Python

Python nyelv

Alapok

Általános jellemzők

- ▶ **A Python programozási nyelv**
 - ▶ egy általános célú
 - ▶ magas szintű
 - ▶ nagyon széles körben elterjedt
 - ▶ ingyenes program nyelv,
- ▶ amely lehetővé teszi a programozás moduláris és objektív megközelítését
- ▶ **Megalkotója**
 - ▶ Guido van Rossum (1989)
 - ▶ azóta számos önkéntessel együtt fejleszti
- ▶ **A név**
 - ▶ Monty Python csoportról
- ▶ **Cél**
 - ▶ olvashatóság
 - ▶ programozói munka megkönnyítése
 - ▶ akár a futási sebesség árán is

Általános jellemzők

- ▶ **Interpretált**
 - ▶ nincs különválasztva a forrás- és tárgykód, a megírt programot sorról-sorra értelmezi és futtatja a Python értelmező (interpreter), pl. CPython
- ▶ **Portábilis**
 - ▶ több (minden?) operációs rendszer és géptípus által támogatott
- ▶ **Sokoldalú**
 - ▶ széles körben alkalmazható a néhányszor tíz soros szkriptektől a több tízezer soros komplex programokig
 - ▶ például: prototípus fejlesztés
- ▶ **Nagyon egyszerű szintaxis**
 - ▶ nagyon tömör, mégis jól olvasható programok írhatóak vele
 - ▶ (azonos funkciójú C/C++/Java program hosszának gyakran csak harmada-ötöde az egyenértékű Python kód)
- ▶ **Dinamikus**
 - ▶ az interpreter ki tud értékelni Python kifejezéseket és utasításokat tartalmazó karakterláncokat

Általános jellemzők

- ▶ **Introspektív**
 - ▶ támogatja a futás közbeni típus és kód ellenőrzést (code/type introspection), emellett számos fejlesztőeszköz pl.: debugger, profiler is magában a Pythonban van implementálva
- ▶ **Dinamikusan és erősen típusos**
 - ▶ Nem használ explicit adattípust változók definiálásánál (szemben a statikusan típusos nyelvekkel), viszont egy változó értékadása után már számít az adott változó típusa
- ▶ **Támogatja a**
 - ▶ komplex struktúrákat nyelvi szinten, a többszálú programozást (multithreading), az objektum orientált programozást, a többszörös öröklést (multiple inheritance), operátor túlterhelést (operator overloading) és virtuális függvényeket (virtual methods)
- ▶ **Bővíthető**
 - ▶ könnyen fejleszthető vagy illeszthető hozzá külső könyvtár (C könyvtár CPython vagy meglévő Java csomagok Jython esetében)
- ▶ **Sokoldalúan támogatott**
 - ▶ számtalan kiegészítő csomag, például stringek, reguláris kifejezések, UNIX szolgáltatások (csővezetékek, socket, ...), internet protokollok (FTP, CGI, HTML, JSON, XML, ...), perzisztencia, adatbázisok, grafikus interfészek...

Különbségek más nyelvektől

- ▶ Utasításblokkok szeparálása behúzással (indentation)
 - ▶ olvashatóság
 - ▶ a jó programok más nyelven is így néznek ki
 - ▶ azonos blokkba tartozó elemek, azonos behúzás (space/tab)
 - ▶ ajánlás: 4db space
- ▶ Üres blokk
 - ▶ kell egy külön kulcsszó: pass
 - ▶ nem csinál semmit, placeholder
 - ▶ pl: egy függvényt majd később írunk meg, de hívni már szeretnénk máshonnan

```
def fib(n):      # Fibonacci sorozat
    a, b = 0, 1
    while a < n:
        print a
        a, b = b, a+b

def test():
    pass
```

Különbségek más nyelvektől

▶ Sorzárás

- ▶ nincs külön karakter, helyette: '\n'
- ▶ többsoros kódnál sortörés: '\n' karakterrel
- ▶ utasítások elválaszthatók: ';' karakterrel

▶ Értékadás

- ▶ <név> = <érték>
- ▶ típus csak futás közben kerül meghatározásra
- ▶ utána már számít, hogy mit tárol (erős típusosság)
- ▶ többszörös értékadás
 - ▶ pl. változók cseréje:
 - ▶ `x, y = y, x`

```
def fib(n):      # Fibonacci sorozat
    a, b = 0, 1
    while a < n:
        print a
        a, b = b, a+b

def test():
    pass
```

Különbségek más nyelvektől

▶ Kommentek

- ▶ ‘#’ karakter után
- ▶ hivatalosan egy soros kommentek

▶ Változók

- ▶ lokális érvényesség
- ▶ globális: ha függvényen kívül definiáljuk
 - ▶ elérhető bárhol
 - ▶ de írás esetén kell a global kulcsszó

▶ Láthatóság, hozzáférhetőség

- ▶ nincs külön nyelvi elem (mint private, protected)
- ▶ helyette: általános névkonvenció
 - ▶ belső használatú függvények, privát metódusok: ‘_’ (aláhúzás) karakterrel kezdődnek
 - ▶ nem “illik” kívülről hozzáférni
 - ▶ de nincs tiltva

```
def fib(n):      # Fibonacci sorozat
    a, b = 0, 1
    while a < n:
        print a
        a, b = b, a+b

def test():
    pass
```

Különbségek más nyelvektől

▶ Minden objektum

- ▶ (mint UNIX esetében: minden fájl)
- ▶ egységes kezelés
- ▶ minden “referálható, azaz változóban tárolható elem” egy objektum,
 - ▶ aminek vannak meghívható metódusai és
 - ▶ amelyekben tetszőleges attribútumokat tárolhatunk el
- ▶ Mik ezek?
 - ▶ osztálpéldányoktól és string változóktól kezdve az egyszerű és komplex típusokon keresztül a különálló függvények és osztálymetódusok, a modulok és a csomagok is!

```
def fib(n):      # Fibonacci sorozat
    a, b = 0, 1
    while a < n:
        print a
        a, b = b, a+b

def test():
    pass
```

Python nyelv

Hello world

Hello world

▶ Első python script

▶ futtatás

- ▶ python test.py
- ▶ vagy futtatható Python script

▶ import

- ▶ külső csomagok beimportálása
- ▶ lehet csak megadott részeket is

▶ def

- ▶ függvény definiálása
- ▶ egy összetett utasítás
- ▶ “:” jelzi, hogy új utasításblokk kezdődik
 - nagyobb behúzás kell
 - itt a függvény törzsének

```
#!/usr/bin/python
import random

def my_main():
    r = "Hello world"
    r = random.randint(0, 10)
    print "Hello random world! ", r

if __name__ == "__main__":
    my_main()
```

Hello world

- ▶ Első python script
 - ▶ értékadás
 - ▶ r változó kap egy string értéket
 - ▶ string
 - ▶ 'asdf' vagy "asdf" (egyenértékűek)
 - ▶ függvényhívás
 - ▶ beimportált random csomag
 - ▶ randint függvényét hívjuk
 - ▶ r változó most egy int értéket kap
 - ▶ print
 - ▶ stdout-ra ír, paraméterként kapott stringeket összefűzve
 - ▶ r változót stringgé konvertálja

```
#!/usr/bin/python
import random

def my_main():
    r = "Hello world"
    r = random.randint(0, 10)
    print "Hello random world! ", r

if __name__ == "__main__":
    my_main()
```

Hello world

▶ Első python script

- ▶ utolsó blokk: biztonsági ellenőrzés
- ▶ if
 - ▶ feltétel vizsgálat
 - ▶ interpreter futtatja-e az aktuális modult/fájlt
 - ha igen, akkor a modulban alapból megtalálható `__name__` paraméter a “`__main__`” stringet tartalmazza
 - egyébként a Python fájl nevét
- ▶ mire jó?
 - ▶ `my_main` függvény csak akkor fut le automatikusan, ha közvetlenül futtatjuk a modult
 - ▶ modul importálásánál nem!

```
#!/usr/bin/python
import random

def my_main():
    r = "Hello world"
    r = random.randint(0, 10)
    print "Hello random world! ", r

if __name__ == "__main__":
    my_main()
```

Python nyelv

Adattípusok

Adattípusok

▶ None

- ▶ nem definiált értékű változó

▶ Numerikus típusok

- ▶ immutable objektumok
- ▶ integer
- ▶ long, float
- ▶ complex

▶ String

- ▶ karakterek sorfolytonos tömbje
- ▶ 'asdf', "asdf"
- ▶ tripla idézőjel (dokumentációhoz)
- ▶ egyszerű műveletek
 - ▶ [:] rész-string kinyerése
 - ▶ + konkatenáció
 - ▶ * ismétlés
 - ▶ [i] adott indexre hivatkozás

```
str = "Hello World!"
```

```
print str           # teljes string -> "Hello World!"
print str[0]        # string első karaktere -> "H"
print str[2:5]      # részstring a 3-6. karakterekből -> "llo "
print str[2:]       # részstring a 3. karaktertől a string végéig -> "llo World!"
print str * 2       # string 2szer -> "Hello World!Hello World!"
print str + "TEST"  # string összefűzés -> "Hello World!TEST"
```

▶ Logikai típus (bool)

- ▶ True, False
- ▶ nagybetűvel!!
- ▶ műveletek
 - ▶ and, or, not
- ▶ más típusú változó is kiértékelhető mint logikai érték
 - ▶ hasonlóan a C-hez

Összetett / Kollekcio típusok

- ▶ tetszőleges típusú értékek!
- ▶ szekvenciatípusok
 - ▶ tárolás sorfolytonos
 - ▶ számít a pozíció
 - ▶ list (lista)
 - ▶ tuple
- ▶ konténertípusok
 - ▶ sorrend nélküli tárolás
 - ▶ dictionary (szótár)
 - ▶ set (halmaz)

Összetett/Kollekció típusok

▶ List (lista)

- ▶ más nyelvek tömbjéhez hasonló
- ▶ elemek sorfolytonos tárolása
- ▶ műveletek
 - ▶ `[a, b]` definiálás
 - ▶ `[i]` adott indexre hivatkozás
 - ▶ `[-j]` negatív index a végéről kezd

▶ műveletek

- ▶ `[:]` rész-lista kinyerése
- ▶ `+` konkatenáció
- ▶ `*` ismétlés
- ▶ `del` elem törlése
- ▶ (hol láttuk ezeket?)

```
my_list = ['abcd', 786 , 2.23, 'john', 70.2]
other_list = ['other', 42]

print my_list           # teljes lista -> [ 'abcd', 786 , 2.23, 'john', 70.2 ]
print my_list[0]        # lista első eleme -> 'abcd'
print my_list[1:3]      # lista 2. 3. 4. eleme egy új listában -> [786 , 2.23, 'john']
print my_list[:2]       # lista minden 2. eleme -> [786 , 'john']
print my_list[2:]       # új lista a régi lista első 2 eleme nélkül -> [2.23, 'john', 70.2]
print my_list.append('13') # elem hozzáfűzése a listához -> ['abcd', 786 , 2.23, 'john', 70.2, 13]
print my_list * 2       # új lista a régi elemeiből 2szer egymás után veve -> ['abcd', 786 , 2.23, 'john', 70.2, 'abcd', 786 , 2.23, 'john', 70.2]
print my_list + other_list # új lista a két listából összefűzve -> ['abcd', 786 , 2.23, 'john', 70.2, 'other', '42']
```

Összetett/Kollekció típusok

- ▶ Dictionary (szótár)
 - ▶ key-value (kulcs-érték) alapú adattárolás
 - ▶ ~ asszociatív tömb, hash tábla
 - ▶ értékek tárolása a kulcs hash-e alapján
 - ▶ gyors kinyerés
 - ▶ DE a sorrend nem definiált!!
- ▶ fontos műveletek
 - ▶ szótár iterálása
 - ▶ dict.keys()
 - ▶ dict.iteritems()

```
my_dict = {'name': 'john', 'code': 6734, 'dept': 'sales'}

my_dict['one'] = "This is one"      # -> {'name': 'john', 'code': 6734, 'dept': 'sales', 'one': "This is one"}
my_dict[2] = "This is two"         # -> {'name': 'john', 'code': 6734, 'dept': 'sales', 'one': "This is one", 2: "This is two"}

del my_dict["name"]                # name kulcsu ertekek torlese -> {'code': 6734, 'dept': 'sales', 'one': "This is one", 2: "This is two"}
print my_dict['one']                # "one" kulcshoz tartozo ertekek -> "This is one"
print my_dict[2]                    # 2 kulcshoz tartozo ertekek -> "This is two"
print my_dict                       # teljes szotar -> {'code': 6734, 'dept': 'sales', 'one': "This is one", 2: "This is two"}
print my_dict.keys()                # szotar kulcsainak tuple-je -> ('code', 'dept', 'one', 2)
print my_dict.values()              # szotart ertekeinek tuple-je -> (6734, 'sales', "This is one", "This is two")
```

Összetett/Kollekció típusok

▶ Tuple

- ▶ kb. csak olvasható lista
- ▶ elemei, mérete nem változtatható
- ▶ használata gyorsabb a listaénál
- ▶ például
 - ▶ `my_tuple = ('abcd', 786 , 2.23, 'john', 70.2)`

▶ Set (halmaz)

- ▶ hasonló, mint a lista, de
- ▶ egy elemet csak egyszer tartalmazhat
- ▶ sorrend nélküli tárolás
- ▶ műveletek
 - halmazműveletek

Python nyelv

Vezérlési szerkezetek

Feltétel vizsgálat

- ▶ Két érték összehasonlítása
- ▶ operátorok
 - ▶ C-ből ismert operátorok
 - ▶ `<`, `>`, `==`, `!=`, `<=`, `>=`
 - ▶ logikai operátorok
 - ▶ `and`, `or`, `not`
 - ▶ “is” kulcsszó
 - ▶ két változó által mutatott objektum megegyezik-e
 - ▶ reference equality
 - ▶ None mindig ugyanarra az objektumra mutat
 - ▶ “in” kulcsszó
 - ▶ tartalmazás vizsgálat

```
var1 = var2 = 12
```

```
var1 is var2           # -> True
```

```
var1 is None           # -> False
```

```
var2 is not None       # -> True
```

```
name = "Peter"
```

```
name in ["John", "Rick"] # -> False
```

```
"e" in name              # -> True
```

Elágazás, while ciklus

▶ Elágazás

- ▶ if-elif-else szerkezet
- ▶ feltétel ellenőrzéshez nem kell ()
- ▶ minden blokk előtt “:”

```
if kifejezés1:  
    kódblokk1  
elif kifejezés2:  
    kódblokk2  
else:  
    kódblokk3
```

▶ while ciklus

- ▶ iterál
- ▶ amíg a megadott feltétel igaz

```
while kifejezés:  
    kódblokk
```

For ciklus

- ▶ Különbözik a megszokottól
- ▶ igazából ~ foreach
- ▶ iterálható objektumon mehetünk vele végig
- ▶ leggyakrabban
 - ▶ összetett típus bejárására
- ▶ enumerate()
 - ▶ list, tuple esetén
 - ▶ (index, érték) párost ad vissza
 - ▶ automatikus felbontás (l, value)
- ▶ iteritems()
 - ▶ dict esetben
 - ▶ minden iterációban (kulcs, érték) párost kapunk

```
for <változó> in <szekvencia>:  
    kódblokk
```

```
for number in (1, 2, 3, 4, 5):    # number = 1..5  
    ...  
  
for key in {1: "a", 2: "b"}:      # key = 1, 2  
    ...
```

```
for i, value in enumerate(('a', 'b', 'c')):    # i = 1..3  value = a, b, c  
    ...  
  
for key, value in {1: 'a', 2: 'b'}.iteritems(): # key = 1, 2  value = a, b  
    ...  
  
for c in "Hello world!":                # c = H, e, l, l, ...  
    ...
```

For ciklus

▶ Hagyományos for ciklus megvalósítása

- ▶ xrange()
- ▶ xrange(30, 100, 3)
 - ▶ lépésköz is megadható

▶ else ág

- ▶ ciklusoknál is van
- ▶ legutolsó szabályos ciklusmag futása után hajtódik végre egyszer
- ▶ ha break-kel lépünk ki
 - ▶ nem fut le

```
for i in xrange(10):  
    ...  
else:  
    # i már elérte a 10et  
  
while i <= 10  
    ...  
else:  
    # i több lett mint tíz
```

Python nyelv

Függvények

Függvények definiálása, hívása

- ▶ def kulcsszó
- ▶ függvény név
- ▶ paraméter lista
- ▶ docstring
- ▶ kódblokk
- ▶ visszatérési érték
 - ▶ tetszőleges típus

```
def func_name([param1, [param2, ...]]):  
    """docstring"""  
    kódblokk  
    return [kifejezés]
```

```
def test1(param1):  
    return param1  
  
def test2():  
    return (1, 2, "kutya")  
  
func_name()  
var1 = test1(13)      # var1 = 13  
a, b, c = test2()     # a = 1, b = 2, c = "kutya"
```

Paraméterek átadása

- ▶ Sajátos, vegyes mechanizmus
 - ▶ immutable objektumok: érték szerint
 - ▶ mutable objektumok: objektum-referencia szerint
 - ▶ ha a függvényen belül megváltoztattjuk, MEGVÁLTOZIK kívül is!

Függvényargumentumok

- ▶ kötelező argumentumok
- ▶ kulcsszó argumentumok
- ▶ alapértelmezett argumentumok
- ▶ változó számú argumentumok
- ▶ ezek vegyíthetők is

```
def func1(param1, param2):
```

```
...
```

```
func1("hello world", 13)    # hivas
```

```
func1(param2 = 13, param1 = "hello world")    # nev szerint hivas
```

```
def func2(a, b, c, d):
```

```
...
```

```
func2(1, 2, d="hello", c = 13)    # vegyes argumentum megadas
```

```
def func3(a, b, c = "default"):
```

```
...
```

```
func3(1,2)
```

Függvényargumentumok

```
def func4(a, b, *args, **kwargs):
```

```
...
```

```
func4(1, 2, 3, 4, x=5, y=6)    # 1, 2 az a,b argumentumokba / 3, 4 az args listaba / 5, 6 az x és y kulccsal a kwargs szotarba csomagolodik be
```

▶ plusz paraméterek

▶ név nélkül

- ▶ `*args` lista argumentumba csomagolódnak

▶ névvel

- ▶ `**kwargs` szótár argumentumba csomagolódnak

```
def test(a, b, c):
```

```
    pass
```

```
params = ("Hello", "World", 42)
```

```
test(*params)
```