Tartalomjegyzék

Bemutatkozás

- Eisenberger András
- Hívjatok Csirkének
- email: csirkeee@gmail.com

Python ismétlés

Aki nem 2014-ben hallgatta az Info1-et és úgy érzi lemaradása van, azok részére elérhet?ek

- Az el?z? félév el?adásainak anyagai: Informatika1-2014/eloadas3, Informatika1-2014/eloadas4
- Vagy akár a hivatalos (angol) Python tanító anyag: Python tutorial.

Mai órán még kb. elég az eloadas3 anyaga, ami kell az függvények definiálása, elágazások.

A Python nagyon népszer? nyelv, többek között jól átlátható szintaxisa miatt, és mivel viszonylag könny? új platformokra is átültetni. Ennek köszönhet?, hogy rengeteg eszköz és er?forrás elérhet? hozzá az interneten. Csak egy példa a http://pythontutor.com, amit itt az el?adás lapjain is használunk a kód részletes bemutatásához, de ti is használhatjátok saját teszteléshez.

Függvényhívás

Python-ban a kódban bárhol definiálható függvény a kódban, a `def` kulcsszóval.

Egyel?re most arról az esetr?l beszélünk csak, amikor a fájl gyökerében van definiálva, nem másik

függvényen belül. Amíg nem tanuljuk meg (kés?bb), hogy miért hasznos függvényen belül függvényt definiálni, addig soha ne tegyetek ilyet:

```
def f1(n):
    ... kód ...
    def f2(i):
        ... kód ...

    ... kód ...
```

Hanem egyel?re semelyik `def` el?tt ne legyen behúzás, a függvények egymás után legyenek definiálva.

Mi történik függvényhíváskor

Itt egy példa függvény:

```
# Függvény, ami a lista minden elemét négyzetre emeli
def negyzetel(L):
    uj_L = []
    for i in L:
        uj_L.append(i*i)
    return uj_L
```

Mi történik egy ilyen függvény meghívásakor? Tehát ha valahol a kódban ilyesmi szerepel, hogy

```
szamok = [5, 1, 8]
negyzetelt_szamok = negyzetel(szamok)
```

Amikor a futtatás a `negyzetel(L)` résznél tart, a következ? dolgok történnek, ilyen sorrendben:

1. A függvénynek a paraméterei megkapják a megfelel? nevüket. (Ebben az esetben az egy paraméter az `L`, annak az értéke az lesz, amit a paraméternek megadtunk, ebben az esetben a `szamok` lista.)
2. A kód futása a függvény els? sorára ugrik.
3. A kód fut tovább amíg a következ? három eset közül valamelyik megtörténik:
    1. Egy `return` parancshoz ér. Ekkor a függvény visszatér a `return` után megadott értékkel.
    2. A futtatás a függvény végére ér, ekkor a függvény `None`-nal tér vissza.
    3. Valamilyen hiba történik, ekkor az egész program leáll.
4. Ha a függvény visszatért, a visszatérési értéke a "http://wiki.math.bme.hufüggvény helyére kerül"http://wiki.math.bme.hu a kódban, ebben az esetben eltároljuk a `negyzetelt_szamok` nev? változóba.

Itt egy kicsit hosszabb példa ami demonstrálja ezeket a lépéseket. Itt két kb. ekvivalens változata látható a kódnak, csak a jobb oldaliban kézzel csinálom meg azokat a dolgokat, amiket a függvény hívás automatikusan:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
    uj_L = []
    for i in L:
        uj_L.append(i*i)
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    elso_szamok = range(1, n+1)
    negyzetszamok = negyzetel(elso_szamok)
```

```
# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    elso_szamok = range(1, n+1)

    # negyzetel függvény végrehajtása
    # paraméter:
    L = elso_szamok
    # kód:
    uj_L = []
    for i in L:
        uj_L.append(i*i)
    # Visszatérés
```

```
    return negyzetszamok                        negyzetszamok = uj_L

                                                return negyzetszamok
```

Ez a kód kipróbálható Python Tutor-on. Itt van egy alternatív, rövidebb változat.

Amiben kicsit csaltam itt, hogy mindennek különböz? nevet adtam, azért lehetett így probléma nélkül beilleszteni a kódot. Még egy fontos dolog, hogy a függvényben mást jelenthetnek a változók. Itt egy példa:

link

Mivel ez zavaró, a legegyszer?bb mód az elkerülésére, ha ti is minden változót különböz?nek neveztek el. Amúgy se árt ha a változók neve elég jól leírja a szerepüket, úgy könnyebb megérteni a kódot olvasás közben.

Függvények és metódusok

A python-ban van két különböz? típusú függvény.

Az egyszer? függvény aminek csak paraméterekre van szüksége, pl. a `sort`:

```
L = [5, 2, 4]
L_2 = sorted(L)
print L_2
```

Itt a `sorted` függvény paramétere a `L` nev? lista, és az alapján el?állítja a sorbarendezett új listát. Maga az `L` lista marad rendezetlen.

A függvények másik típusának hivatalos neve "http://wiki.math.bme.humetódus"http://wiki.math.bme.hu. Ezek olyan függvények amiket egy változón hajtunk végre. Úgy néz ki, hogy a változó neve után teszünk egy pontot, majd utána jön a függvény neve és paraméterei. Erre egy példa a `sort`:

```
L = [5, 2, 4]
L.sort()
print L
```

Ezt úgy is fogalmazhatjuk, hogy az `L` lista `sort` metódusát hívjuk meg. Metódusoknak is lehetnek még paraméterei, bizonyos szempontból olyan mintha a változó amin meghívtuk lenne az els? paraméter. Ebben az esetben ez a metódus módosítja az `L` listát, és nem egy új listád ad vissza.

Általában jellemz?bb hogy a változó metódusai módosítják a változót, míg amikor paraméterként adjuk át a változót, akkor valószín?leg nem módosul. Ez nem törvényszer?, de általánosságban lehet így gondolni rájuk. Vannak más, gyakorlatibb különbségek is, de azokkal most még nem foglalkozunk.

Hibakeresés

Amikor a Labor feladatokat és a Házi feladatokat próbáljátok megoldani CloudCoder-en, hasznos, ha magatoktól meg tudjátok találni a hibákat a kódban. Ehhez próbálok most egy kis segítséget adni.

A következ? kód ugyanazt a problémát próbálja megoldani mint az el?bb, hogy adjuk vissza az els? n négyzetszámot, de van vele néhány probléma:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
```

```
    for i in L:
        uj_L.append(L[i]**L[i])
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    elso_szamok = range(1, n)
    negyzetszamok = negyzetel(elso_szamok)
    return negyzetszamok
```

Lehet hogy néhány problémát els?re látsz, de amikor a saját kódodról van szó, akkor ez kevésbé igaz, úgyhogy végigmegyünk hogy lépésenként hogy lehet megtalálni a hibákat.

Két fajta hiba van alapvet?en, olyan hogy le sem tud sikeresen futni a program, és olyan, amikor ugyan lefut, de rossz eredményt ad. Az el?bbi esetben a CloudCoder nem mutatja meg a publikus teszteseteket se meg, úgyhogy ha ez történik, akkor el?ször azt kell megoldani. Egy egyszer? és gyors megoldás ha a megoldás függvény elejére beírjuk hogy `return`. Így a függvény mindjárt visszatér, ugyan semmilyen értéket nem ad vissza, de ez már sikeres lefutásnak számít. Tehát ekkor így néz ki a kód:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
    for i in L:
        uj_L.append(L[i]**L[i])
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    return
    elso_szamok = range(1, n)
    negyzetszamok = negyzetel(elso_szamok)
    return negyzetszamok
```

Így ebben az esetben pl. megtudom, hogy az els? tesztesetnél a bemenet `4`, és a várt kimenet `[1, 4, 9, 16]`. Ezek után több lehet?ségem van a hiba megkeresésére, nézzük sorban.

Python Tutor

Ha hiba van, a Python Tutor pirossal jelzi, így annak a forrását viszonylag könny? megtalálni. De a hiba lekövetésénél is jól jöhet hogy minden változó értékét mutatja folyamatosan. Itt vannak ennek a kódnak a hibakeresésének különböz? fázisai:

Kiindulás

1. lépés

2. lépés

3. lépés

Hibátlan kód

Ez nagyon könnyen használható és jó módszer, a nagy hátránya hogy maximum 300 lépést enged, ami már akár bonyolultabb feladatoknál itt is lehet hogy nem lesz elég.

print-elés nagyban

Ehhez a módszerhez jó ha a saját gépen is tudtok python kódot futtatni. Ehhez windows-on van egy kis segítség itt, a linux legtöbb változatán alapból fel van rakva a python és futtatható.

Tehát, tudva a példa inputot, kiegészítem a kódot egy teszttel:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
    for i in L:
        uj_L.append(L[i]**L[i])
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    elso_szamok = range(1, n)
    negyzetszamok = negyzetel(elso_szamok)
    return negyzetszamok

print elso_negyzetszamok(4)
```

Ezt a kódot elmentem egy fájlba, majd próbálom futtatni, a következ? kimenetet kapom a python-tól:

```
Traceback (most recent call last):
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz.py"http://wiki.math.
    print elso_negyzetszamok(4)
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz.py"http://wiki.math.
    negyzetszamok = negyzetel(elso_szamok)
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz.py"http://wiki.math.
    uj_L.append(L[i]*L[i])
NameError: global name 'uj_L' is not defined
```

Ilyen több lépéses hibaüzenetnél mindig a legalsó blokkot kell nézni, az mondja meg, hogy a tényleges hiba a kódom 5. sorában szerepelt, ami úgy néz ki, hogy `uj_L.append(L[i]*L[i])`, és a hiba az volt, hogy az `uj_L` nev? változó nincs definiálva. Ezt a hibát kijavítva, más más hibát kapunk, amit kicsit talán nehezebb értelmezni:

```
Traceback (most recent call last):
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz_2.py"http://wiki.mat
    print elso_negyzetszamok(4)
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz_2.py"http://wiki.mat
    negyzetszamok = negyzetel(elso_szamok)
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz_2.py"http://wiki.mat
    uj_L.append(L[i]**L[i])
IndexError: list index out of range
```

De az kiderül, hogy a hiba a 6. sorban van, ami már nagy segítség. Ha az adott sorban több minden is történik, akkor érdemes szétrobbantani azt a sort több kisebb részre, és azokat vizsgálni. A python olyan programnyelv, hogy ha hiba van, akkor is a hibáig elfut, és csak ott adja fel, ezért erre kit?n?en használható a `print` parancs. Itt van a kód egy már jelent?sen kisegített változata ehhez az esethez:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
    uj_L = []
    for i in L:
        print "i:   ", i
        print "L:   ", L
```

```
        print "L[i]:", L[i]
        uj_L.append(i**i)
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    elso_szamok = range(1, n)
    negyzetszamok = negyzetel(elso_szamok)
    return negyzetszamok

print elso_negyzetszamok(4)
```

Így a kimenetünk a következ?

```
i:    1
L:    [1, 2, 3]
L[i]: 2
i:    2
L:    [1, 2, 3]
L[i]: 3
i:    3
L:    [1, 2, 3]
L[i]:
Traceback (most recent call last):
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz_2.py"http://wiki.mat
    print elso_negyzetszamok(4)
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz_2.py"http://wiki.mat
    negyzetszamok = negyzetel(elso_szamok)
  File "http://wiki.math.bme.huD:\egyeb\Dropbox\Info2\eloadas\negyzetel_rossz_2.py"http://wiki.mat
    print "http://wiki.math.bme.huL[i]:"http://wiki.math.bme.hu, L[i]
IndexError: list index out of range
```

Így már az `L[i]` sornál baj van, és rájöhetünk hogy az `i` már maga a lista eleme, nincs szükség az `L[i]`-re.
Így eljutunk odáig, hogy a program sikeresen lefut, csak rossz eredményt ad, ekkor minden lépést kiírva megkereshetjük hol van el?ször probléma:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
    print "negyzetel fv, bemenet", L
    uj_L = []
    for i in L:
        print "i =", i, ", i**i =", i**i
        uj_L.append(i**i)
    print "negyzetel fv, kimenet", uj_L
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    print "n", n
    elso_szamok = range(1, n)
    print "elso_szamok", elso_szamok
    negyzetszamok = negyzetel(elso_szamok)
    print "negyzetszamok", negyzetszamok
    return negyzetszamok

print elso_negyzetszamok(4)
```

Ekkor a kimenetünk:

```
n 4
elso_szamok [1, 2, 3]
```

```
negyzetel fv, bemenet [1, 2, 3]
i = 1 , i**i = 1
i = 2 , i**i = 4
i = 3 , i**i = 27
negyzetel fv, kimenet [1, 4, 27]
negyzetszamok [1, 4, 27]
[1, 4, 27]
```

Innen már a maradék hibákat akár láthatjuk is, hogy már az `elso_szamok` lista túl rövid, és hogy a négyzetre emelésnél `i*i` helyett `i**i`-t írtunk, ami az `i`-edik hatványra emelt. Ezeket kijavítva jó a megoldás, ekkor kitörölhetjük az immár felesleges `print`-eket.

Interaktív mód használata

Tegyük fel hogy odáig már eljutottunk, hogy a program sikeresen lefut, tehát jelenleg ez a kódunk:

```
# Függvény, ami a lista minden
# elemét négyzetre emeli
def negyzetel(L):
    uj_L = []
    for i in L:
        uj_L.append(i**i)
    return uj_L

# Az els? n négyzetszám listában
def elso_negyzetszamok(n):
    elso_szamok = range(1, n)
    negyzetszamok = negyzetel(elso_szamok)
    return negyzetszamok
```

Ezeket a függvényeket a python interaktív módjában is lehet tesztelni. Elindítjuk a parancssorból a python-t, majd simán bemásoljuk a kódot, ekkor a függvények hívhatóak.

```
>>> negyzetel([3, 1, 4])
[27, 1, 256]
>>> negyzetel([3])
[27]
>>> 3**3
27
```

Ekkor rájövünk hogy itt mi a baj, kijavítjuk azt:

```
def negyzetel(L):
    uj_L = []
    for i in L:
        uj_L.append(i*i)
    return uj_L
```

Berakjuk ezt a változatot az interaktív python-ba, majd:

```
>>> negyzetel([3, 1, 4])
[9, 1, 16]
>>> range(1, 4)
[1, 2, 3]
>>> range(4)
[0, 1, 2, 3]
>>> range(1, 5)
[1, 2, 3, 4]
>>> n = 4
>>> range(1, n+1)
```

```
[1, 2, 3, 4]
```

Így itt is kikisérletezzük hogy hogy lehet jól megoldani, és akkor ezt berakjuk a kódba:

```
def elso_negyzetszamok(n):
    elso_szamok = range(1, n+1)
    negyzetszamok = negyzetel(elso_szamok)
    return negyzetszamok
```

És ha berakjuk interaktív módba akkor tesztelhetjük is:

```
>>> elso_negyzetszamok(5)
[1, 4, 9, 16, 25]
```

Összegzés

Remélhet?leg ezek a módszerek hasznosak számotokra. Ez persze nem azt jelenti, hogy minden hibát magatoktól meg kell találni, a Laboron nyugodtan szóljatok ha bármi gond van, de pl. amikor a házit írjátok, nem leszünk ott hogy segítsünk, ezért szerettem volna plusz eszközöket tenni a kezeitekbe.