Bevezetés a Python programozási nyelvbe

Szathmáry László
Debreceni Egyetem
Informatikai Kar

11. Gyakorlat

- JSON szerializáció (folyt.)
- reguláris kifejezések
- Függelékek; összefoglaló

(utolsó módosítás: 2018. aug. 28.)

JSON (folyt.)

Átkerült az **S)** függelékbe…

Reguláris kifejezések

Egy mintát (egy reguláris kifejezést) keresünk egy nagyobb sztringben.

```
import re

re.search(pat, text)
```

általános alakja

```
>>> import re
>>>
>>> match = re.search('fej', 'regularis kifejezesek')
>>>
>>> match
<_sre.SRE_Match object at 0x7fd93cabe2a0>
>>>
>>> match.group()
'fej'
>>>
>>> match = re.search('unix', 'regularis kifejezesek')
>>> print match
None
>>> match.group()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'NoneType' object has no attribute 'group'
```

a visszatérési értéke egy match objektum

az a rész, amire illeszkedett a minta

sikertelen keresés esetén a visszatérési érték: None

Mivel a re.search visszatérési értéke vagy egy match objektum vagy pedig egy None érték, ezért tipikusan a következőképp szokás használni:

```
match = re.search(pat, text)
if match:
    # volt találat…
```

Teszteléshez használjuk ezt az egyszerű kis függvényt:

```
 9  def Find(pat, text):
10      match = re.search(pat, text)
11      if match:
12          print match.group()
13      else:
14          print 'nincs benne'
```

```
Find('fej', 'reguláris kifejezések')
```

benne van

```
Find('unix', 'reguláris kifejezések')
```

nincs benne

```
>>> Find(r'reg\.', 'reg. kif.-ek')
reg.
```

„raw” sztring: úgy veszi a sztringet, ahogy leírtuk.
A \ jelnek nincs speciális jelentése ekkor, ugyanolyan karakter, mint a többi.

A reguláris kifejezés (minta) megadásánál érdemes mindig használni!

Reguláris kifejezés alapminták

- `a`, `X`, `3` (hagyományos karakterek, önmagukra illeszkednek)
  Speciális karakterek: `. ^ $ * + ? { } [ ] ( ) \ |` (ezek nem illeszkednek önmagukra, különleges jelentéssel bírnak)
- `.` (bármilyen karakter, kivéve az újsort (‘`\n`’))
- `\w` (Szó karakter: `[a-zA-Z0-9_]`. Csak egy karakterre illeszkedik, nem egy egész szóra!)
- `\W` (nem-szó karakter)
- `\b` (boundary, határ szó és nem-szó között)
- `\s`, `\S` (whitespace és nem-whitespace)
- `\d`, `\D` (számjegy (`[0-9]`) és nem-számjegy)
- `\t`, `\n`, `\r` (tab, újsor, return)
- `^`, `$` (sor eleje, sor vége)
- `\` (Az őt követő karakter speciális jelentését elveszi. Pl. `\.` ténylegesen a pontot jelenti, a `\$` ténylegesen a `$` jelet, stb.)

Egy sztringből szedjük ki az email címet:

```
>>> Find(r'[\w.]+@[\w.]+', 'vmi jabba.laci@gmail.com vmi')
jabba.laci@gmail.com
```

Szedjük ki a felhasználó illetve a host nevét külön-külön:

```
>>> m = re.search(r'([\w.]+)@([\w.]+)', 'vmi jabba.laci@gmail.com vmi')
>>> m
<_sre.SRE_Match object at 0x2053470>
>>> m.group()
'jabba.laci@gmail.com'
>>> m.group(1)
'jabba.laci'
>>> m.group(2)
'gmail.com'
```

A zárójelek használata NEM változtatja meg a keresési mintát, csupán bizonyos részeket elment, megjegyez.

re.findall

A `re.search` után ismerkedjünk meg a `re.findall` függvénnyel:

```
>>> s = 'vmi jabba.laci@gmail.com valami foo@bar'
>>> re.findall(r'[\w.]+@[\w.]+', s)
['jabba.laci@gmail.com', 'foo@bar']
```

Nem áll meg az első találatnál, hanem megy tovább s **az összes egyezést** visszaadja egy listában.

Mi a helyzet zárójelek használata esetén?

```
>>> s = 'vmi jabba.laci@gmail.com valami foo@bar'
>>> re.findall(r'([\w.]+)@([\w.]+)', s)
[('jabba.laci', 'gmail.com'), ('foo', 'bar')]
```

Tuple-ök listája. Egy tuple egy egyezés. A tuple elemei a bezárójelezett részek.

```
>>> dir(re)
['DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M', 'MULTILINE', …
>>>
>>> s = 'vmi jabba.laci@gmail.com VMi foo@bar'
>>> re.findall(r'vmi', s, re.IGNORECASE)
['vmi', 'VMi']
```

3. paraméter
pl. a kis- és nagybetűket ne különböztesse meg

Greedy vs. non-greedy

```
>>> html = '<b>foo</b>  <i>bar</i>'
>>>
>>> re.search(r'<.*>', html).group()
'<b>foo</b>  <i>bar</i>'
>>>
>>> re.search(r'<.*?>', html).group()
'<b>'
```

mohó

nem mohó

Csere: re.sub

`re.sub(mit, mire, text)`

Használható a `\1`, `\2`, …, ami a `group(1)`, `group(2)`, … -re utal.

```
>>> s = 'vmi jabba@gmail.com valami leia@gmail.com'
>>> print re.sub(r'([\w.]+)@[\w.]+', r'\1@tatooine.com', s)
vmi jabba@tatooine.com valami leia@tatooine.com
```

Az email host részét lecseréli.

`re.sub(mit, mire, text, **szám**)`

Alapértelmezésben az **összes** előfordulást lecseréli.

Ha megadjuk a **szám** paramétert, akkor **csak ennyi előfordulást** cserél.

Függelékek

Ideje megnézni a Függelékeket is…

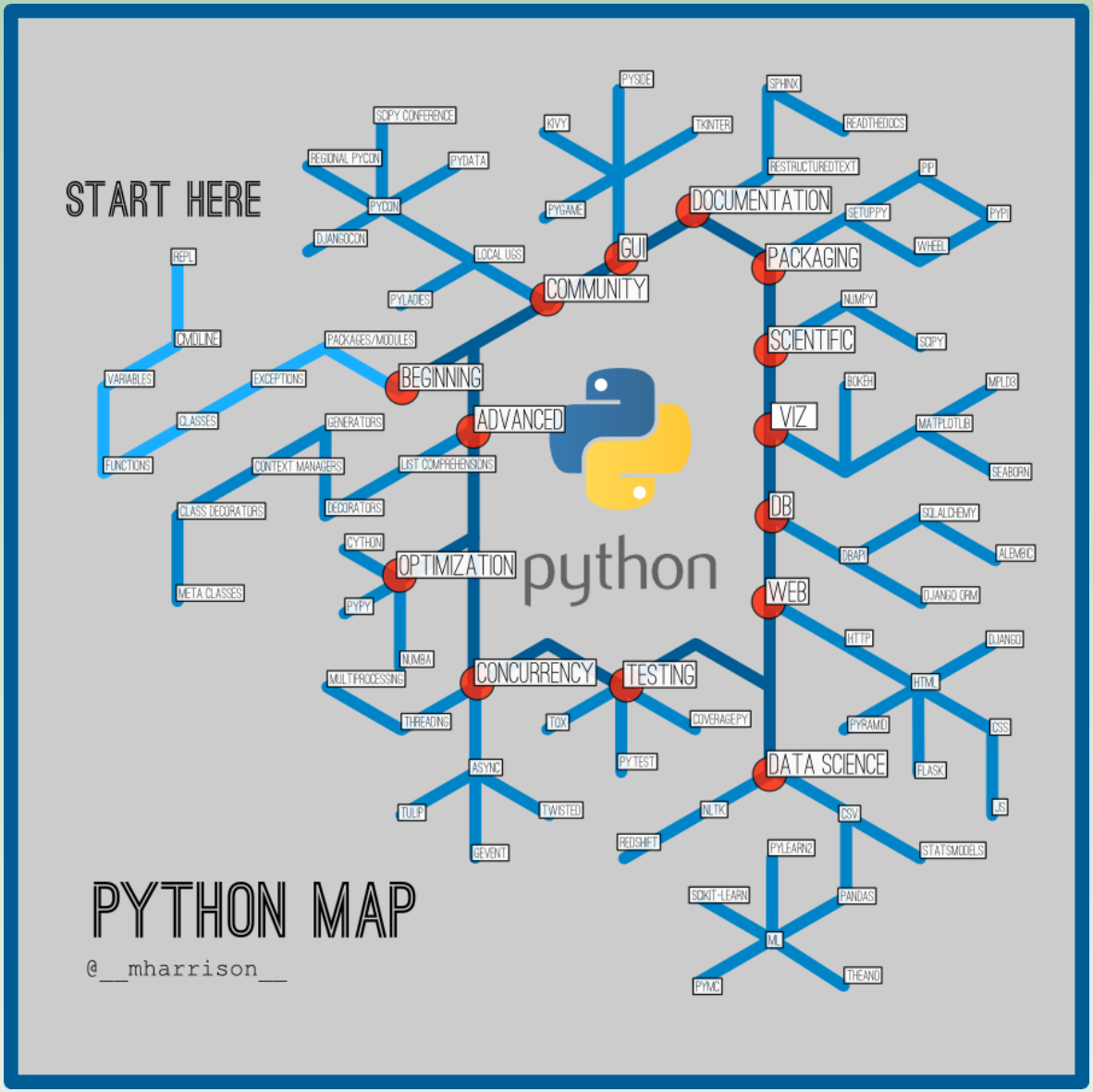
Hogyan tovább?

- *Guido van Rossum*: **Python Tutorial** (http://docs.python.org/tutorial/index.html, PDF)
- *Wesley J. Chun*: **Core Python Programming** (2nd Edition)
- *Doug Hellmann*: **The Python Standard Library by Example** (Developer's Library)
  [online változat: **Python Module of the Week** (https://pymotw.com/3/)]

Ismerjék meg a standard library-t!

Hogyan tovább?

(link)



Haladó témák

- osztályok, OO programozás
- web scraping
- GUI fejlesztés Qt-vel
- web programozás (Flask)
- adatbáziskezelés (SQLite), SQLAlchemy (ORM)
- kapcsolat az operációs rendszerrel
- processzek, szálak
- unit tesztek
- iterátorok, generátorok, dekorátorok
- …