Programska potpora komunikacijskim sustavima

Dr. sc. Adrian Satja Kurdija

Programski jezik Python - 5. predavanje

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Sadržaj predavanja

- Iteratori
- Iznimke
- Serijalizacija
- Interakcija s operacijskim sustavom
- HTTP zahtjevi

Iteratori

Iteratori

- Iteratori se koriste u for petljama i komprehenzijama
- Iterator predstavlja strukturu od više elemenata koju je moguće obići element po element
- Liste, skupovi, n-torke i mape su iteratori (između ostalog)
- Ako nam klasa predstavlja niz podataka koji želimo obići npr. ovako:
  `for artikl in moj_racun: ...`
  ... onda u klasi treba definirati metode `__iter__` i `__next__`
- Metoda `__iter__` stvara iterator
  - obično inicijalizira neki brojač i vraća `self`
- Metoda `__next__` vraća idući element
  - npr. `self.artikli[self.brojac]`

Iteratori

- Kako radi for petlja?
- Kad napišemo:

```
for i in range(3): ...
```

... u pozadini se poziva:

```
it = range(3).__iter__()          # ili it = iter(range(3))
i = it.__next__() --> vraća 0     # ili i = next(it)
i = it.__next__() --> vraća 1
i = it.__next__() --> vraća 2
i = it.__next__() --> vraća iznimku StopIteration
```

Klasa kao iterator

- Primjer - Fibonaccijevi brojevi:

```
class Fib:
  def __init__(self, max):
    self.max = max

  def __iter__(self):
    self.a = 1
    self.b = 1
    return self

  def __next__(self):
    fib = self.a
    if fib > self.max:
      raise StopIteration
    self.a, self.b = self.b, self.a + self.b
    return fib
```

```
for x in Fib(100):
    print(x)
```

Ispis:

1 1 2 3 5 8 13 21 34 55 89

Klasa kao iterator

- **Zadatak**
- Omogućiti iteriranje po zaposlenicima poduzeća iz prethodnog zadatka, tako da je moguće izvesti npr.:

```
for zaposlenik in moje_poduzece:
    print(zaposlenik)
```

Iznimke

Iznimke

- Iznimke su sintaksne, semantičke ili namjerne greške prilikom izvođenja programa
- Npr. dijeljenje nulom, korištenje nepostojećeg objekta ili datoteke, pristup nepostojećem elementu niza ili rječnika...
- To su klase koje nasljeđuju klasu *Exception*
- Po defaultu uzrokuju prekid izvršavanja i ispis odgovarajuće poruke:

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

`try` i `except` blokovi

- Ako ne želimo prekid izvršavanja, iznimku možemo prepoznati unutar bloka `try:`

```
while True:
    try:
        x = int(input("Unesite broj: "))
        break
    except ValueError:
        print("Neispravan broj! Pokusajte ponovno.")
```

- Možemo prepoznati više vrsta iznimki:

```
except (RuntimeError, TypeError, NameError):
    pass  # ignoriranje iznimke - ne preporučuje se!
```

Višestruki `except` blokovi

```
import sys

try:
  f = open('myfile.txt')
  s = f.readline()
  i = int(s.strip())
except IOError as err:
  print('I/O error: {0}'.format(err))
except ValueError:
  print('Could not convert data to an integer.')
except:
  print('Unexpected error:', sys.exc_info())
```

Uzrokovanje iznimki

- Naredba raise izaziva bilo koju iznimku:
```
>>> raise NameError('HiThere')
```
- Možemo definirati vlastite iznimke:

```
class MyError(Exception):
  def __init__(self, value):
    self.value = value
  def __str__(self):
    return "MyError with value " + str(self.value)

try:
    raise MyError(5)
except MyError as e:
    print(e)
```

Iznimke - vježba

- Definirati tip iznimke `ErrZap`
  - inicijalno bez ikakvih metoda (npr. `pass` umjesto tijela klase)
- Proširiti klasu `Poduzece` tako da baca iznimku `ErrZap` ako se:
  - dodaje zaposlenik koji je već u poduzeću
  - izbacuje zaposlenik koji nije u poduzeću
- Napisati klase `ErrZapDodaj` i `ErrZapIzbaci` koje nasljeđuju `ErrZap` i omogućuju razlikovanje gornjih dvaju slučajeva
  - Implementirati metode `__init__` i `__str__` radi ispisa poruke o detaljima greške
  - npr. `Greska pri izbacivanju: Ana Anic ne postoji!`
- "Uhvatiti" iznimke u glavnom programu (`try...except`)

Serijalizacija

Serijalizacija: `pickle`

- *Pickling* – serijalizacija Python objekta u string ili niz bajtova
- *Unpickling* – suprotna operacija

```
import pickle

>>> pickle.dumps((1,2))
'(I1\nI2\ntp0\n.'

>>> pickled = pickle.dumps((1,2))
>>> pickle.loads(pickled)
(1, 2)
```

Serijalizacija: `pickle`

- Binarna serijalizacija u datoteku (`dump`, `load`) koristi se unutar odgovarajućeg bloka:

```
with open(filename, "rb | wb") as file:
    pickle.dump(obj, file)    # za pisanje, ili
    obj = pickle.load(file)   # za citanje
```

- **Zadatak**: serijalizirati neki objekt klase `Poduzece` u datoteku. Potom u drugom programu učitati i ispisati serijalizirani objekt iz datoteke.

Interakcija s operacijskim sustavom

Modul `os`

- Portabilno sučelje prema operacijskom sustavu
- Primjer: traženje i brisanje datoteke u direktoriju:

```
import os

def findfile(start, name):
  for dirpath, dirs, files in os.walk(start):
    if name in files:
      full_path = os.path.join(dirpath, name)
      os.remove(full_path)
```

Modul `os`

- Izvršavanje naredbe kao iz terminala:

```
os.system("firefox skripta.pdf")
```

- Tako možemo pokrenuti bilo koji drugi program

- Fleksibilnija alternativa: **`subprocess.run`**

```
subprocess.run(["firefox", "skripta.pdf"])
subprocess.run(
     ["/usr/bin/git", "commit", "-m",
     "Fixes a bug."])
```

Direktoriji i datoteke

`os.getcwd()`
putanja trenutnog direktorija

`os.chdir(dirname), os.chdir('..')`
navigacija: cd, cd ..

`os.listdir()`
lista - sadržaj direktorija: dir (win), ls (unix)

`os.mkdir(path)`
novi direktorij

`os.rename(old_path, new_path)`
preimenovanje ili premještanje

`os.path.basename(filepath)`
vraća ime datoteke bez putanje

`os.path.splitext(filepath)`
odvaja ekstenziju, vraća (file, ext) npr. `('/home/datoteka', '.txt')`

Modul `os`: vježba

- Napisati program koji:
  - pronalazi abecedno prvu datoteku u zadanom direktoriju,
  - stvara novi poddirektorij `tmp`,
  - premješta datoteku u `tmp` ali joj mijenja ime u `prva` (zadržava ekstenziju).
- Npr. datoteka `/home/adrian/Desktop/fotka.jpg` postaje `/home/adrian/Desktop/tmp/prva.jpg`
- Koristiti:
  - `os.listdir(path)`
  - `os.path.isdir(filepath)`
  - `os.path.basename(filepath), os.path.splitext(filepath)`
  - `os.mkdir(path)`
  - `os.path.join(path, file)`
  - `os.rename(old, new)`

HTTP zahtjevi

HTTP zahtjevi

- Modul **`requests`**
- Pojednostavljuje slanje HTTP zahtjeva
  - zaglavlja (*headers*), forme, datoteke
  - parametri su mape (*dict*)
  - povratne vrijednosti su također Python objekti
- Korištenje:
  - `pip install` **`requests`**
  - `import requests`
  - `r = requests.get(...)`
  - `r = requests.post(...)  # put, delete, ...`
  - `r` je `Response` objekt
    - `r.status_code, r.content, r.text, r.headers`

HTTP zahtjevi: GET

```
r = requests.get('https://google.hr')
print(r.text)

query = {'lat': '45', 'lon': '180'}
r = requests.get(
        'http://api.open-notify.org/iss-pass.json',
        params=query)
print(r.headers)
print(r.json())

url = 'https://api.github.com/some/endpoint'
r = requests.get(url,
        headers={'user-agent': 'my-app/0.0.1'})
```

HTTP zahtjevi: POST

```
>>> payload = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.post("https://httpbin.org/post",
                      data=payload)
>>> r.text
{
  ...
  "form": {
    "key2": "value2",
    "key1": "value1"
  },
  ...
}
```

HTTP zahtjevi: POST

```
>>> url = 'https://httpbin.org/post'
>>> files = {'file': open('report.xls', 'rb')}
>>> r = requests.post(url, files=files)
>>> r.text
{
  ...
  "files": {
    "file": "<censored...binary...data>"
  },
  ...
}
```

HTTP zahtjevi

- Bad requests

```
bad_r = requests.get('http://httpbin.org/status/404’)
>>> bad_r.status_code
404
>>> bad_r.raise_for_status()
requests.exceptions.HTTPError: 404 Client Error
```

HTTP zahtjevi: vježba

- Napisati program koji (uz pomoć GET zahtjeva) ispisuje imena astronauta koji se trenutačno nalaze u ISS
- API: http://api.open-notify.org/astros.json
  - koristiti `response.json()` kao mapu (dict)
  - dokumentacija: http://open-notify.org/Open-Notify-API/People-In-Space/