# Delo s podatki v programskem jeziku Python

Splošnonamenski programski jezik **Python** je trenutno **najpopularnejši jezik** zaradi enostavne sintakse in obilice prosto-dostopnih programskih knjižnic, dočim pa ni najhitrejši jezik. Jezik se interpretira, kar pomeni, da program `demo.py` v ukazni vrstici izvedemo s `python demo.py`.

## Programske knjižnice in skripte

V programskem jeziku **Python knjižnice in skripte** uvozimo z ukazom `import`. Knjižnice je potrebno uvoziti pred uporabo, dočim obstajajo različne oblike uporabe ukaza `import`.

```python
import demo
import requests
import requests as req
from requests import * # ni priporočljivo
```

## Programske zbirke podatkov

V programskem jeziku **Python nabor** določimo z običajnimi oklepaji `(...)` ali konstruktorjem `tuple()` pri čimer ni potrebno, da so elementi nabora enakega tipa. Nabor je **urejena nesprejemljiva zbirka** podatkov kar pomeni, da *ne moremo* dodajati ali brisati elementov.

```python
t = (0, 1, 'foo', 'bar')
st = t[:2]
ln = len(t)
```

V programskem jeziku **Python seznam** določimo z oglatimi oklepaji `[...]` ali konstruktorjem `list()` pri čimer ni potrebno, da so elementi seznama enakega tipa. Seznam je **urejena sprejemljiva zbirka** podatkov kar pomeni, da *lahko* dodajamo ali brišemo elemente po vrednosti in indeksu.

```python
l = [0, 1, 'foo', 'bar']
sl = l[:2]
ln = len(l)
l[1] = -1
del l[0]
l.remove(-1) # zbriše prvo pojavitev
l.append(9.81)
l.extend([0, 'baz'])
```

V programskem jeziku **Python množico** določimo z zavitimi oklepaji `{...}` ali konstruktorjem `set()` pri čimer ni potrebno, da so elementi množice enakega tipa. Množica je **neurejena sprejemljiva zbirka** podatkov kar pomeni, da *lahko* dodajamo ali brišemo elemente po vrednosti.

```python
s = {0, 1, 'foo', 'bar'}
ln = len(s)
s.remove(1) # zbriše edino pojavitev
s.add(9.81)
s.update([0, 'baz'])
```

V programskem jeziku **Python slovar** določimo z zavitimi oklepaji `{...}` ali konstruktorjem `dict()` pri čimer ni potrebno, da so ključi ali vrednosti slovarja enakega tipa. Slovar je **neurejena sprejemljiva zbirka** podatkov kar pomeni, da *lahko* dodajamo ali brišemo vrednosti po ključu.

```python
d = {0: 'foo', 'bar': 1}
ln = len(d)
d[1] = 'baz'
d.pop('bar') # zbriše edino pojavitev
d.update({'bar': 1})
l = dict.values() # iterator vrednosti
l = dict.keys() # iterator ključev
l = dict.items() # iterator parov
```

V programskem jeziku **Python zbirko uredimo** z uporabo metode `sort` ali funkcije `sorted`. Slednja vrne urejen seznam elementov, dočim morajo biti elementi enakega tipa. Na drugi strani pa metoda `shuffle` ustvari naključno permutacijo elementov seznama, dočim ni potrebno, da so elementi enakega tipa.

```python
s = {'foo', 'bar', 'baz'}
l = sorted(s) # ustvari nov seznam
l.sort() # urejanje na mestu

import random
random.shuffle(l)
```

Kaj vrneta funkciji `iter` in `enumerate`, če ju uporabimo nad Python zbirko? Slednja se izkaže kot uporabno, ko iteriramo čez elemente zbirke in hkrati potrebujemo indeks (tj. zaporedno številko) elementa.

V spodnji tabeli so prikazane **časovne zahtevnosti osnovnih operacij** nad Python zbirkami, kjer je $n$ število elementov zbirke in $i$ indeks (tj. zaporedna številka) elementa.

| Zbirka | Konstruktor | Iskanje | Vstavljanje | Brisanje | Urejanje | Duplikati? |
|---|---|---|---|---|---|---|
| nabor | `()` / `tuple()` | $\mathcal{O}(n)$ | / | / | / | ja |
| seznam | `[]` / `list()` | $\mathcal{O}(n)$ | $\mathcal{O}(n-i)$ | $\mathcal{O}(n-i)$ | $\mathcal{O}(n \log n)$ | ja |
| množica | `{}` / `set()` | $\approx \mathcal{O}(1)$ | $\approx \mathcal{O}(1)$ | $\approx \mathcal{O}(1)$ | / | ne |
| slovar | `{}` / `dict()` | $\approx \mathcal{O}(1)$ | $\approx \mathcal{O}(1)$ | $\approx \mathcal{O}(1)$ | / | ne |

## Branje podatkov iz datoteke

V programskem jeziku **Python datoteko odpremo** s funkcijo `open`, **beremo** z uporabo funkcij `read` ali `readline` in **zapremo** z metodo `close`. Pri tem je priporočena uporaba programskega konstrukta `with open(..., 'r') as ...`, ki po koncu branja samodejno zapre datoteko. Vsebino datoteke lahko preberemo v celoti, dočim navadno beremo zaporedoma po vrsticah.

```python
file = open('file.txt', 'r')
print(file.readline())
print(file.read())
file.close()

file = open('file.txt', 'r')
for line in file:
  print(line)
file.close()

with open('file.txt', 'r') as file:
    for line in file:
        print(line)
```

## Pisanje podatkov v datoteko

V programskem jeziku **Python datoteko odpremo** s funkcijo `open`, **pišemo** z uporabo metode `write` in **zapremo** z metodo `close`. Pri tem je priporočena uporaba programskega konstrukta `with open(..., 'w'|'a') as ...`, ki po koncu pisanja samodejno zapre datoteko. Vsebino datoteke lahko zapišemo v celoti, dočim navadno pišemo zaporedoma po vrsticah.

Python
```
file = open('file.txt', 'w')
file.write('line\nline\n')
file.write('line\n')
file.close()

with open('file.txt', 'w') as file:
    for i in range(10):
        file.write('{:d}. line\n'.format(i + 1)) # formatiran izpis
```

## Luščenje vsebine spletnih strani

V programskem jeziku **Python spletno stran** preberemo z uporabo programskih knjižnic `http.client`, `requests` ali drugih. Pri tem spletno stran vedno preberemo v celoti, dočim lahko naknadno iteriramo po vsebini spletne strani z uporabo programskih zank.

Primer uporabe **Python knjižnice** **`http.client`** je prikazan spodaj.

Python
```
import http.client
conn = http.client.HTTPSConnection('urnik.fmf.uni-lj.si')
conn.request('GET', '/layer_one/44/')
text = conn.getresponse().read().decode()
print(text)
```

Primera uporabe **Python knjižnice** **`requests`** je prikazan spodaj.

Python
```
import requests
req = requests.get('https://urnik.fmf.uni-lj.si/layer_one/44/')
text = req.text # HTML format
print(text)

req = requests.get('http://ip.jsontest.com/')
json = req.json() # JSON format
print(json)
```

## Razčlenjevanje nizov z regularnimi izrazi

V programskem jeziku **Python regularne izraze** uporabljamo za razpoznavanje, razčlenjevanje in iskanje nizov znakov. Regularni izraz predstavlja želeni oziroma iskani vzorec znakov, ki ga definiramo kot `r'...'`. Pri tem lahko uporabljamo rezervirane znake oziroma vzorce naštete spodaj.

- `.` predstavlja poljuben znak razen nove vrstice
- `\d` predstavlja poljubno števko ali cifro
- `\D` predstavlja poljubno neštevko in necifro
- `\w` predstavlja poljuben alfanumerični znak
- `\W` predstavlja poljuben nealfanumerični znak
- `\s` predstavlja poljuben beli znak (npr. presledek)
- `\S` predstavlja poljuben nebeli znak (npr. števko)
- `\` omogoča iskanje ubežnih znakov (npr. `r'\.'`)
- `^` predstavlja začetek niza znakov ali vrstice
- `$` predstavlja konec niza znakov ali vrstice

(Rezervirane) znake oziroma vzorce lahko združujemo, ponavljamo in gnezdimo kot je našteto spodaj.

- `(...)` predstavlja zaporedje znakov (npr. `r'(abc)'`)
- `[...]` predstavlja množico znakov (npr. `r'[a-zčšžA-ZČŠŽ]'`)
- `[^...]` predstavlja negacijo množice znakov (npr. `r'[^a-c]'`)
- `|` predstavlja disjunkcijo znakov (npr. `r'\d|[a-c]'`)
- `*` predstavlja nič ali več ponovitev vzorca (npr. `r'\d*'`)
- `+` predstavlja eno ali več ponovitev vzorca (npr. `r'[abc]+'`)
- `?` predstavlja največ eno ponovitev vzorca (npr. `r'[a-c]?'`)
- `{n}` predstavlja natanko $n$ ponovitev vzorca (npr. `r'[a-c]{3}'`)
- `{n,m}` predstavlja med $n$ in $m$ ponovitev vzorca (npr. `r'[a-c]{1,3}'`)

Pri delu z regularnimi izrazi navadno uporabljamo **Python knjižnico `re`** (https://docs.python.org/3.9/library/re.html).

**Funkcija `match`** preveri ali *začetek* niza znakov ustreza podanemu regularnemu izrazu. Funkcija vrne `None`, če se niz ne začne z regularnim izrazom, sicer pa objekt razreda `Match`, ki vrne ujemanje z uporabo funkcije `group`.

Python
```
import re
string = '-123.45'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.match(regex, string)
if res == None:
    print('Not integer!')
else:
    print(res.group())
```

**Funkcija `search`** preveri ali niz znakov *vsebuje* podani regularni izraz. Funkcija vrne `None`, če niz ne

vsebuje regularnega izraza, sicer pa objekt razreda `Match`, ki vrne ujemanje z uporabo funkcije `group`.

```python
import re
string = 'Is this integer -123?'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.search(regex, string)
if res == None:
    print('No integer found!')
else:
    print(res.group())
```

**Funkcija** `findall` poišče vse *pojavitve* podanega regularnega izraza v nizu znakov. Funkcija vrne seznam ujemanj regularnega izraza, ki je lahko prazen.

```python
import re
string = 'Find all integers in 123.45!'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.findall(regex, string)
for i in res:
    print(i)
```

**Funkcija** `finditer` poišče vse *pojavitve* podanega regularnega izraza v nizu znakov. Funkcija vrne seznam objektov razreda `Match`, ki je lahko prazen.

```python
import re
string = 'Find indices of integers in 123.45!'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.finditer(regex, string)
for i in res:
    print(i.start(), i.end())
```

**Funkcija** `split` *razbije* niz znakov glede na podan regularni izraz. Funkcija vrne seznam razbitja niza znakov, ki je lahko prazen.

```python
import re
string = 'Split by integers -123 and 45!'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.split(regex, string)
for str in res:
    print(str)
```

**Funkcija** `sub` *zamenja* vse pojavitve podanega regularnega izraza v nizu znakov. Pri tem lahko

zamenjavo določimo kot niz znakov…

```python
import re
string = 'Replace integers 123 and 45!'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.sub(regex, '<int>', string)
print(res)
```

…ali pa zamenjavo določimo z uporabo podane lambda funkcije.

```python
import re
string = 'Replace integers -123 and 45!'
regex = r'[+-]?[1-9][0-9]*|0'
res = re.sub(regex, lambda res: 'x' * len(res.group()), string)
print(res)
```