# Objektno programiranje
# ⟨ Upravljanje tijekom programa, formatirani ispis, nizovi (stringovi, liste, n-terci), rječnici, skupovi ⟩

Tihomir Žilić, Mario Essert, Vladimir Milić

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

Zagreb, 2021./2022.

# Sadržaj

# Upravljanje programom

- Programsko upravljanje je redoslijed po kojem se programski kôd izvršava. Upravljanje tijekom programa u svim programskim jezicima, temelji se na uvjetnim naredbama, petljama i pozivima funkcija.
- Često se neka naredba ili niz (blok) naredbi treba izvršiti samo u slučaju ako je neki uvjet zadovoljen. Ponekad je to izvršavanje ovisno o nekoliko međusobno povezanih uvjeta. Kontrola tijeka `if`, koja koristi proširenja `elif` ili `else` zaključak, služi za takvo uvjetno izvršavanje naredbi.
- Petljom `for` ostvaruje se iterativno izvršavanje naredbe ili bloka naredbi, a upravlja se preko iteracijskog izraza. Koristi se kada je broj iteracija unaprijed poznat.
- Petljom `while` izvršava se naredbu ili blok naredbi niti jedan, jedan ili više puta, ovisno o ispunjenju upravljanog uvjetnog izraza. Može se shvatiti kao ponavljajuća `if` kontrola tijeka.

# if-elif-else kontrola tijeka programa

- Pythonova sintaksa je sljedeća:

```
if izraz:
    naredbe
elif izraz:
    naredbe
else:
    naredbe
```

- izraz predstavlja jedan ili više uvjeta definiranih relacijskim (==, !=, >, itd.) operatorima i povezanih logičkim (and, or, not) operatorima.
- Svaki rezultat iz izraz izračunava se kao istinit (True) ako nije prazan. Inače, rezultat iz izraz izračunava se kao neistinit (False).

# for petlja

- Pythonova sintaksa je sljedeća:

```
for iteracijska_varijabla in iteracijski_niz:
    naredbe
```

- Naredba for slijedno re-povezuje iteracijska_varijabla na svaki element po redu unutar iteracijski_niz. Naredba ili naredbe od kojih se sastoji tijelo petlje izvršavaju se jedna po jedna za svaki element.
- Naredbe unutar tijela petlje se ne izvršavaju ako iteracijski_niz ne oslobađa nikakve elemente. U tom slučaju, iteracijska_varijabla se ne povezuje na niti jedan način pomoću naredbe for.
- Ako iteracijski_niz oslobađa barem jedan element, kad se naredbe petlje završe, iteracijska_varijabla ostaje povezana na posljednju vrijednost na koju ju je naredba petlje povezala.
- for petlja u Pythonu se može koristiti i u sažetom obliku:

```
[izraz for iter_var in iter_niz odlomci]
```

# while petlja

- Pythonova sintaksa je sljedeća:

```
while izraz:
      naredbe
```

- izraz predstavlja jedan ili više uvjeta definiranih relacijskim i logičkim operatorima.
- Ako izraz nije istinit (False), niti jedna od naredbi u tijelu petlje se ne izvršava. Ako je izraz istinit (True), naredbe se izvršavaju.
- Kada se naredbe izvrše, uvjet se ponovno izračunava, da se vidi treba li se izvršiti nova iteracija. Ovaj se proces nastavlja sve dok izraz ne postane neistinit, nakon čega petlja završava. Tijelo petlje treba sadržavati dio koji će u nekom trenutku učiniti izraz neistinitim, ili petlja nikada neće završiti, osim ako se ne izvrši naredbu break.

# Naredbe break, continue i pass

- `break` - iskače iz (unutarnje) petlje.
- `continue` - skače na novu iteraciju petlje.
- `pass` - je nul-operator, tj. ništa se ne događa kada se izvrši.

# Formatiranje ispisa, `print()`

Ispis pomoću funkcije `print()`: na zaslon računala, u datoteku, na izlaznu napravu. Funkcija `print()` koristi se za ispis stringa:

1. na zaslon računala koristeći:

   string nadovezu elemenata odvojenih zarezom, ili povezanih plusom:

```
>>> print("Ovo","je",1,"auto")
>>> print("Ovo " + "je " + "1 " + "auto")
Ovo je 1 auto
Ovo je 1 auto
```

   formatirajući string operator %, npr.:

```
>>> print("Danas je %s i %d. dan!" % ("lijep", 8))
Danas je lijep i 8. dan!
```

   string metodu `format()`, npr.:

```
>>> print("{0} s vrijednostima {1}".format("Lista", [1,2,3]))
Lista s vrijednostima [1, 2, 3]
```

2. u datoteku na disku:

```
>>> moja_dat = open('datoteka.txt', 'w')
>>> print('Upisao sam tekst u datoteka.txt!', file = moja_dat)
>>> moja_dat.close()
```

# Spremnici, engl. *containers*

- Ugrađeni Pythonovi spremnici su: `list`, `tuple`, `dict`, `set`. Ostali posebni spremnici su dio modula `collections`.
- Spremnici se pri pisanju kôda definiraju preko objekata, ali u memoriji spremnici sadrže samo identitete na te objekte.
- Nizovi: spremnici s poredanim (sekvencijalnog tipa, iterabilni) objektima[1]:
  - **Lista**, `[O1, O2, ..., On]`, *npr.* `['bca', 3.1, 5+3j]`
  - **N-terac**, `(O1, O2, ..., On)`, *npr.* `("12+3j", 'liste', 7)`
  - **String**, *npr.* `'neki string'`
- Spremnici s neporedanim objektima:
  - **Rječnik**, {`k1:v1, k2:v2, ..., kn:vn`}, *npr.* {`'auto':'zelen', 'gume':4`}
  - **Skup**, `set((LO1, LO2, ..., LOn))`, *npr.* `set((1,7,5,2+5j,'6'))`

[1] `O` ≡ objekt, `k` ≡ ključ elementa, `v` ≡ vrijednost elementa, `LO` ≡ literal objekt

Količina identiteta (adresa) unutar spremnika:

- promjenjiva: lista, rječnik i skup,
- nepromjenjiva: string, n-terac, *frozen* skup.

# Indeksiranje objekata

**NIZ**

indeksi → | 0 | 1 | 2 | n |

niz = | $id_0$ | $id_1$ | $id_2$ | $id_n$ |

niz[0] = | $id_0$ |

niz[n] = | $id_n$ |

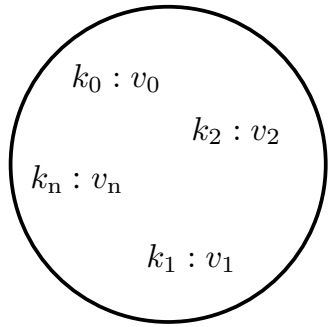
niz[0:3] = | $id_0$ | $id_1$ | $id_2$ |

niz[1:] = | $id_1$ | $id_2$ | $id_n$ |

niz[-1] = | $id_n$ |

**RJEČNIK**

rjec =

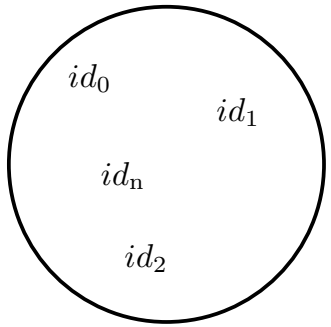



rjec[$k_1$] → $v_1$
rjec[$k_n$] → $v_n$

id, k, v su identiteti objekata
k:v označuju parove

**SKUP**

S =




S[?]

# Lista - poredani spremnik identiteta (adresa objekata)

- Engl. *list*.
- Objekti odvojeni zarezom unutar uglatih zagrada `[O1, O2, ...]`, ili unutar funkcije `list((O1, O2, ...))`.
- Bilo koji tip objekta, npr.:
  - `[373]`, `[373,]`, `[ ]`
  - `['Što je to?', 3.14, ["ugnježđena lista", False, True, 45], 2]`
  - `[2.2, [1,2,3], {'A':4,'B':5,'C':'6'}, 5]`
  - `list((2.2, [1,2,3], {'A':4,'B':5,'C':'6'}, 5))`
- **Promjenjivi** su identiteti objekata unutar liste i njihova količina.

# Lista - poredani spremnik identiteta objekata

Promjenjivi su identiteti unutar liste i njihova količina:

```
>>> # PROMJENA VELICINE LISTE (kolicina identiteta)
>>> f=[1,3+4j,[1,2],'abc']  # lista s identitetom u f
>>> id(f)
3064466764
>>> f.append(67)     # dodavanje novog objekta u listu
>>> f
[1, (3+4j), [1, 2], 'abc', 67]
>>> id(f)
3064466764
>>> g=f     # stvaranje nove varijable g s istom adresom kao f
>>> id(g)
3064466764
>>> g.append('kraj') # dodavanje novog objekta u listu
>>> g
[1, (3+4j), [1, 2], 'abc', 67, 'kraj']
>>> f
[1, (3+4j), [1, 2], 'abc', 67, 'kraj']
>>> # PROMJENA identiteta unutar liste
>>> f[0]='pocetak'  # promjena prvog identiteta unutar liste
>>> f
['pocetak', (3+4j), [1, 2], 'abc', 67, 'kraj']
>>> id(f)
3064466764
```

# Lista - kopiranje (**shallow copy**)

*Shallow copy* - kopiranje identiteta objekata liste na drugu memor. lokaciju
Funkcija copy() iz modula copy ili :

```
>>> z=g[:]    # kopiranje liste (shallow copy) ili >>> z=list(g)
>>> id(z)
3064411308
>>> g.append(11)
>>> f
['pocetak', (3+4j), [1, 2], 'abc', 67, 'kraj', 11]
>>> g
['pocetak', (3+4j), [1, 2], 'abc', 67, 'kraj', 11]
>>> z
['pocetak', (3+4j), [1, 2], 'abc', 67, 'kraj']
>>> f[2].append(7) # dodavanje 7 u ugnjezdenu listu
>>> f
['pocetak', (3+4j), [1, 2, 7], 'abc', 67, 'kraj', 11]
>>> g
['pocetak', (3+4j), [1, 2, 7], 'abc', 67, 'kraj', 11]
>>> z
['pocetak', (3+4j), [1, 2, 7], 'abc', 67, 'kraj']
```

# Lista - kopiranje (**deepcopy**)

Kopiranje objekata liste, funkcija *deepcopy()* iz modula *copy*.
Ako je neki od kopiranih objekata promjenjivog tip (lista, skup, rječnik), kopiraju se cijeli objekti na nove memorijske lokacije, inače se kopiraju samo identiteti.

```
>>> from copy import deepcopy  # ucitavanje funkcije iz modula
>>> dc=deepcopy(f)   # kopiranje vrijednosti svih elemenata liste
>>> f[2].append(3.14) # dodavanje 3.14 u podlistu
>>> f
['pocetak', (3+4j), [1, 2, 7, 3.14], 'abc', 67, 'kraj', 11]
>>> g
['pocetak', (3+4j), [1, 2, 7, 3.14], 'abc', 67, 'kraj', 11]
>>> z
['pocetak', (3+4j), [1, 2, 7, 3.14], 'abc', 67, 'kraj']
>>> dc
['pocetak', (3+4j), [1, 2, 7], 'abc', 67, 'kraj', 11]
```

# N-terac - poredani spremnik identiteta objekata

- Engl. *tuple*.
- Objekti odvojeni zarezom unutar oblih zagrada `(O1 ,O2, ...)`, ili unutar funkcije `tuple(O1, O2, ...)`.
- bilo koji tip objekta, npr.:
  - `(373,)`, `( )` - **NAPOMENA:** npr. `(373)` nije *tuple* nego cijeli broj 373 napisan unutar zagrada!
  - `('Što je to?', 3.14, ["ugniježđena lista", False, True, 45], 2)`
  - `(2.2, [1,2,3], {'A':4,'B':5,'C':'6'}, (5,7))`
  - `tuple((2.2, [1,2,3], {'A':4,'B':5,'C':'6'}, (5,7)))`
- **Nepromjenjivi** su identiteti unutar n-terca kao i veličina n-terca (količina identiteta).
- Slijedi da je vrijednost n-terca **nepromjenjiva** ako su objekti literali, a **promjenjiva** ako su objekti promjenjivi poput liste/rječnika/skupa.

# N-terac - poredani spremnik identiteta objekata

```
>>> # PROMJENA VELICINE N-TERCA (kolicina identiteta)
>>> f=(1,3+4j,[1,2],'abc')  #stvaranje n-terca
>>> id(f)
3064133628
>>> f[4]=55   # dodavanje novog objekta u n-terac
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # PROMJENA VRIJEDNOSTI N-TERCA
>>> f[1]=4  # mijenjanje drugog objekta podliste
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> f[2].append('dodano') # dodavanje novog objekta u podlistu
>>> id(f)
3064133628
>>> f[2][1]=22
>>> id(f)
3064133628
>>> f[2].insert(0,7)  # mijenjanje prvog objekta podliste
>>> f
(1, (3+4j), [7, 1, 22, 'dodano'], 'abc')
>>> id(f)
3064133628
```

# Rječnik - neporedani spremnik identiteta objekata

- Engl. *dictionary*.
- Elementi[2] **ključ:vrijednost** odvojeni zarezom unutar vitičastih zagrada `{k1:v1,k2:v2,...}`, ili unutar funkcije `dict(k1:v1,k2:v2,...)`
- **Ključ**evi, različiti, nepromjenjivog tipa (int, float, string, tuple, bool).
- **Vrijednost** može biti bilo koji tip objekta, npr.:
  - `{'a':373}`, `{'a':373,}`, `{ }`
  - `{'float':2.2, 'lista':[1,2,3], 'rjecnik':{'A':4,'B':5,'C':'6'}, 'n-terac':(5,7)}`
  - `{'a':'Što je to?', (30,12):3.14, 7.77:["ugnježđena lista", False, True, 45], 12:2, True:False, 2+3j:7}`
  - `dict(dva=2,R=4)`, ali `dict(3=2,R=4)` !
- **Promjenjive** su i veličina rječnika (količina elemenata) i vrijednost rječnika (vrijednosti pojedinačnih objekata).

[2] Svaki element unutar rječnika definira se u formi *objekt:objekt*, pri čemu se lijevi objekt naziva *ključ* elementa, a desni *vrijednost* elementa.

# Skup - neporedan spremnik adresa nepromjenjivih objekata

- Engl. *set*.
- Funkcija `set( )`, koja ima jedan argument (listu, N-terac ili rječnik, a čiji su elementi nepromjenjivi objekti - znači NE listama i rječnicima).
  Valjani skupovi: `set([1,'3', 2+3j, (1,2)])`, `set((1,'3'))`, `set({'a':1,'re':5, 're':6})`
  dok:

```
set((1,'3',2+3j,(1,2),[5,5]))
TypeError: unhashable type: 'list'
```

- Ne sadrži duplikate, kod rječnika uzima samo ključeve.
- Neporedan skup i nema ključeva, zato se ne može indeksirati.
- **Promjenjiva** je veličina skupa (količina identiteta) ali ne i vrijednosti objekata, metoda `add()` dodaje elemente.
- Smrznuti skup (engl. *frozenset*) ima i **nepromjenjivu veličinu skupa**, tj. ovaj skup uopće nema metodu `add()`!

# String - poredani niz znakova

- `'Jednostruki', '"Dvostruki", '''Trostruki navodnici(""" """) '''`
- `\n, \t, \", npr. print("Prebaci navodnike \n \" u novi red i \t uvuci")`
- Nadovezivanje, npr. `"Jedan" + 'dva' + '3'`, ali `"Jedan" + 'dva' + 3` vraća `TypeError: can only concatenate str (not "int") to str`.
- Generiranje, npr. `"Jedan"+'dva'*3`.

# String - formatiranje

- Formatiranje stringa pomoću metode `format()`:

```
>>> ime='Nada'; godine=100;
>>> 'Ime joj je {0} i ima {1} godina!'.format(ime, godine)
'Ime joj je Nada i ima 100 godina!'
```

- Formatiranje stringa pomoću `f"..."`:

```
>>> ime='Nada'; godine=100;
>>> f'Ime joj je {ime} i ima {godine} godina!'  #od verzije 3.6
'Ime joj je Nada i ima 100 godina!'
```

- Formatiranje string pomoću operatora %, npr.:

```
>>> "Danas je %s i %d. dan!" % ("lijep",8)
Danas je lijep i 8. dan!
```

# String - poredani niz znakova

- **Nepromjenjive** i veličina i vrijednost pri istoj adresi stringa:

```
>>> # PROMJENA VRIJEDNOSTI
>>> A="neki string"     # osnovni string s adresom u A
>>> A
'neki string'
>>> B=A.capitalize()    # promjena vrijednost od A
>>> B          # novostvoreni string s adresom u B
'Neki string'
>>> id(A)    #adresa stringa "neki string"
3068750528
>>> id(B)    # adresa stringa 'Neki string'
3068750648
>>> # PROMJENA VELICINE
>>> A[12]='Z'
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> C=A+"3"    # novostvoreni string s adresom u C
>>> C
'neki string3'
>>> id(C)  # adresa stringa 'neki string3'
3064199912
```