

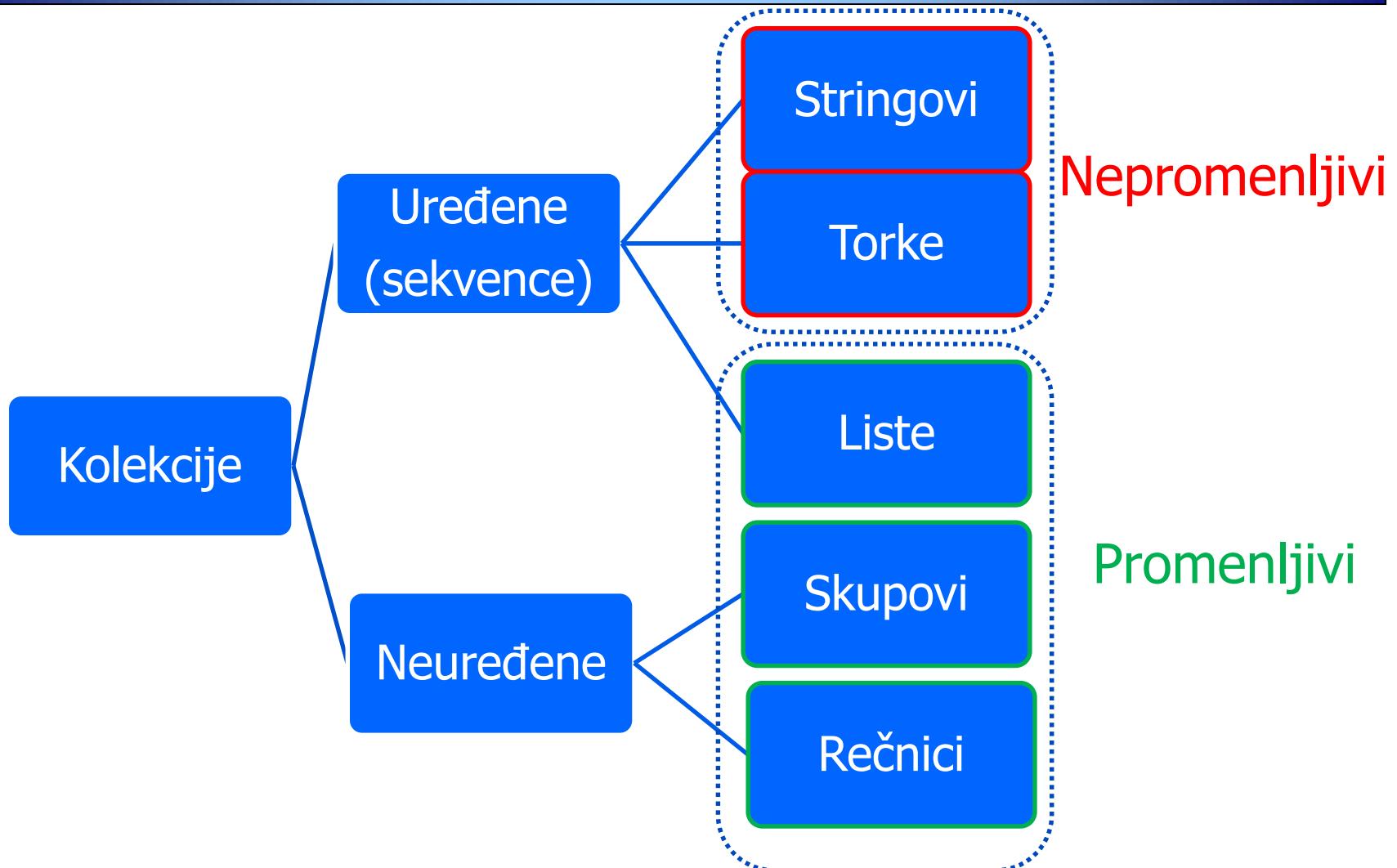
# Programiranje 1

## Kolekcije objekata na programskom jeziku Python

Univerzitet u Beogradu  
Elektrotehnički fakultet  
2019/2020.

# Sekvence, skupovi i rečnici

# Kolekcije objekata - rekapitulacija



# Stringovi – tip `str`

---

- Tekstualne sekvence predstavljaju tekstualni objekat tipa `str`
- Moguće je kreirati ga dodelom vrednosti
- Tekst u istom redu okružuje se apostrofima ili navodnicima
- Tekst u više redova se okružuje trostrukim apostrofima ili navodnicima
- Može se kreirati i konstruktorom `str()`
- Na primer `str(3.14)` daje '`3.14`', a `str(True)` daje '`True`'

# Komponente stringa

---

- String je sačinjen od znakova (karaktera)
- Karakter predstavlja pisani simbol koji ima svoju numeričku (binarnu) reprezentaciju u memoriji
  - Binarna reprezentacija zavisi od sistema kodiranja
- Osnovni ASCII sistem kodira znakove sa 7 bita
  - Sadrži 128 znakova
  - Prošireni ASCII kodira sa 8 bita, odnosno sadrži 256 znakova
- Python koristi Unicode standard
  - Standard određuje skup znakova koji se koriste i bazično ih numeriše 16-bitno
- Za samo kodiraje koristi se UTF-8 sistem
  - Svaki karakter predstavlja sa 8 do 32 bita

# Komponente stringa (1)

---

- Python prepoznaje jedinstveni kodni broj u Unicode sistemu
  - *Unicode Code Point (UCP)*
- *UCP* se dobija pomoću funkcije `ord(znak)`
- Sam znak se dobija pomoću `chr(UCP_broj)`
- Dužina stringa dobija se pomoću `len(string)`
- Python nema poseban tip za pojedinačne karaktere, kao što u nekim drugim jezicima postoji tip `char`
  - Karakter se tretira kao string dužine 1

# Znakovi

---

```
>>> ord('A')  
65  
>>> chr(65)  
'A'  
>>> ord('ч')  
1095  
>>> chr(1096)  
'҆'  
>> len("Alfanumerik")  
11
```

Važi:

**ord(chr(broj)) == broj**  
**chr(ord(znak)) == znak**

# Komponente stringa (2)

---

- Znakovi (komponente) stringa se indeksiraju od 0 do n-1, ako je n dužina stringa
- Kao i kod liste, poslednji element se može indeksirati sa -1, a negativni indeksi koriste se na isti način
  - Dozvoljen *slicing*  
`>>> b = "Hello, World!"`  
`>>> print(b[-5:-2])`  
or  
`>>> print(b[1:10:2])`  
`el,Wr`
- Pristup elementu koji ne postoji (npr. `b[100]`) prijavljuje grešku  
`IndexError: string index out of range`

# Delovi stringa - *slicing*

---

- Nema funkcije za obrtanje stringa
- Postoji mogućnost da se redosled elemenata obrne pomocu *slicing-a*

```
>>> b = "ANA VOLI MILOVANA"  
>>> b = b[::-1]  
>>> print(b)  
ANAVOLIM ILOV ANA
```

- Objašnjenje:
  - U opštem obliku za slicing se koristi **[start, stop, step]**
  - Ako se granice ne navedu: ceo niz
  - Ako je step negativan, kreće se od kraja

# Delovi stringa – *slicing* (negativan korak)

---

```
a[:]                      # ceo string  
a[start:stop:step]        # sa korakom step  
a[-1]                     # poslednji element  
a[-2:]                    # poslednja dva  
a[::-1]                    # svi, obrnuto  
a[1::-1]                  # prva dva, obrnuto  
a[:-3:-1]                 # poslednja dva, obrnuto  
a[-3::-1]                 # sve sem poslednja 2,  
                          # obrnuto  
a[:-2]                     # sve sem poslednja 2  
# kada je korak negativan, od kraja ka početku!
```

# Relacione operacije

---

- Na dva stringa moguće je primeniti relacione operatore
  - `==` i `!=` porede stringove znak po znak
  - `>`, `<`, `>=`, `<=` koriste leksikografski poredak: redom poređi odgovarajuće znakove iz dve sekvene po rezultatu metode `ord(znak)`
  - Ukoliko su znaci jednaki, a u sledećem koraku se jedan string iscrpi, on je manji

```
>>> print ("a"<"b", "aaa"<"b", "aa"<"aab")
True True True
```

# Pripadnost i nadovezivanje (1)

---

- Može se proveriti pripadnost znaka ili podstringa korišćenjem operatora `in`:

```
>>> poruka = "Srećna Nova godina!"
```

```
>>> print ("a" in poruka, "ova" in poruka)  
True True
```

- Može se raditi nadovezivanje korišćenjem operatora `+`:

```
>>> poruka = "Srećni" + "praznici!"
```

```
>>> print (poruka)  
Srećni praznici!
```

- Može se raditi množenje korišćenjem operatora `*`:

```
>>> "Živeli" + 5*"!"
```

```
Živeli!!!!
```

# Prirodnost i nadovezivanje (2)

---

- Postoji i operator `not in`

```
>>> poruka = "Srećna Nova godina!"
```

```
>>> print ("Z" not in poruka)
```

True

- Nadovezivanje korišćenjem operatora `+=`:

```
>>> zdravica += poruka
```

```
>>> print (zdravica)
```

Živeli!!!! Srećna Nova godina!

- Prefiks `r` služi da se *escape* sekvenca doslovno tumači kao tekst

- Bez `r` je `\n` specijalna sekvenca za prelazak u novi red

```
>>> print(r"\n")
```

\n

# Nepromenljivost stringa

---

- Neophodno je naglasiti da je string nepromenljiv tip!

```
>>> niz = "Babe"
```

```
>>> niz[0] = "ž"
```

```
# generiše se greska...
```

```
TypeError: 'str' object does not support item
```

```
>>> id(niz)
```

```
1630486115312
```

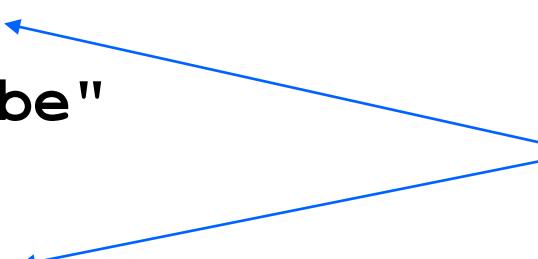
```
>>> niz+=" i žabe"
```

```
>>> id(niz)
```

```
1630486144176
```

```
# Formiran je novi string "Babe i žabe" !
```

Različiti objekti!



# Neke metode za rad sa stringom (1)

Metoda	Opis
<code>t.lower()</code>	Vraća novi string, pretvara sva slova u mala
<code>t.upper()</code>	Vraća novi string, pretvara sva slova u velika
<code>t.swapcase()</code>	Vraća novi string, mala slova pretvori u velika, a velika u mala
<code>t.strip()</code> <code>t.lstrip()</code> <code>t.rstrip()</code>	Vraća novi string bez belih znakova na početku i kraju stringa. Prefiks <code>l</code> samo na početku, prefiks <code>r</code> samo na kraju.
<code>t.count(t1)</code>	Vraća broj pojavljivanja podstringa <code>t1</code> u stringu <code>t</code>
<code>t.index(t1)</code> <code>t.rindex(t1)</code>	Vraća indeks prvog pojavljivanja podstringa <code>t1</code> u <code>t</code> . Prefiks <code>r</code> – poslednje pojavljivanje.

# Neke metode za rad sa stringom (2)

Metoda	Opis
<code>t.find(t1)</code> <code>t.rfind(t1)</code>	Vraća indeks prvog pojavljivanja podstringa <code>t1</code> u <code>t</code> . Vraća <code>-1</code> ako ga nema u <code>t</code> . Prefiks <code>r</code> – poslednje pojavljivanje.
<code>t.startswith(t1)</code> <code>t.endswith(t1)</code>	Vraća <code>True</code> ukoliko string <code>t</code> počinje prefiksom <code>t1</code> . Vraća <code>True</code> ukoliko se string <code>t</code> završava prefiksom <code>t1</code> .
<code>t.replace(ts, tp)</code>	Vraća novi tekst sa zamenjenim <code>ts</code> sa <code>tp</code>
<code>t.join(iterable)</code>	Vraća string koji konkatenira elemente iz <code>iterable</code> sa separatorom <code>t</code> ("-.join("123")) daje 1-2-3
<code>t.split([sep])</code>	Vraća listu reči na osnovu delimitera <code>sep</code> .
<code>t.partition(sep)</code>	Vraća 3-tuple gde je prva komponenta string pre <code>sep</code> , druga sam <code>sep</code> , a treća string iza <code>sep</code> . Ako ne nađe <code>sep</code> , prva je ceo string, a druga i treća su prazne.

# String – primer provere palindroma (1)

---

- Pronalaženje svih palindroma u stringu
- Najpre je data iterativna verzija:

```
def palindromI(tekst):  
    ''' da li je tekst palindrom '''  
    n = len(tekst)  
    limit = n//2  
    for i in range(limit):  
        if tekst[i] != tekst[-i-1]:  
            return False  
    return True
```

# String – primer provere palindroma (2)

---

- Zatim je data i rekurzivna verzija:

```
def palindromR(tekst):  
    ''' da li je tekst palindrom '''  
    if tekst == '':  
        return True  
    elif tekst[0] != tekst[-1]:  
        return False  
    else:  
        return palindromR(tekst[1 : -1])
```

# String – primer provere palindroma (3)

---

- Primer korišćenja:

```
tekst = input('tekst? ')  
n = len(tekst)  
for duzina in range(n, 0, -1):  
    for pocetak in range(0, n-duzina+1):  
        podtekst = tekst[pocetak : \  
                           pocetak+duzina]  
        if palindromR(podtekst):  
            print(podtekst)
```

# Formatiranje teksta (1)

---

- Koristi se metoda `format()` string objekta
- Raspoređuje argumente na mestima vitičastih zagrada
  - Redom ili po brojevima (indeksima) u zagradama
  - Prvom argumentu odgovara redni broj 0

```
>>> '{} unija {}'.format('A', 'B')
```

```
'A unija B'
```

```
>>> 'pi = {}'.format(3.14)
```

```
'pi = 3.14'
```

```
>>> '{2}-{1}-{0}'.format('I', 'II', 'III')
```

```
'III-II-I'
```

# Formatiranje teksta (2)

---

- Moguće je zadati željenu širinu polja pri ispisu
- Prilikom ispisa se sadržaj uravnava
  - Kod tekstualnih podataka koristi se levo uravnavanje
  - Kod numeričkih podata koristi se desno uravnanje
  - Moguće je promeniti pomoću > odnosno <
- Za centriranje u polju date širine koristi se ^

```
>>> '{:5} je {:>5}'.format("Rim", "lep")
```

```
'Rim je lep'
```

```
>>> '{1:5} je {0:>5}'.format("Rim", "lep")
```

```
'lep je Rim'
```

```
>>> 'Taj{:^9}!'.format("Rim")
```

```
'Taj Rim ! '
```

# Formatiranje teksta (3)

---

- Kada je širina manja od potrebne ignoriše se
- Kada se stavi `.širina`, dolazi do odsecanja

```
>>> 'Srećan {:.3}!'.format("putnik")
'Srećan putnik!'
```

```
>>> 'Srećan {:.3}!'.format("putnik")
'Srećan put!'
```

- Za cele, realne i brojeve u eksponencijalnoj notaciji mogu se koristiti specifikatori formata `d`, `f` i `e`
  - Uz dodatak prefiksa za kontrolu ispisa
- Zadaje se širina polja za ispis, broj decimala, eventualni prefiks `+` (znak) ili `0` (popunjavanje nulama umesto blanko znacima)

# Formatiranje teksta (4)

---

```
>>> pi = 3.14
>>> 'pi = {:.3f}'.format(3.14)
'pi = 3.140'
>>> 'x = {:06d}'.format(30)
'x = 000030'
>>> 'x = {:+6d}'.format(30)
'x = +30'
>>> 'f = {:.2f}'.format(1/3)
'f = 0.33'
>>> 'f = {:.4.2e}'.format(1/6)
'f = 1.67e-01'
```

# Kreiranje liste prema pravilu (1)

---

- Programski jezik Python poseduje posebnu notaciju za stvaranje liste prema unapred zadatom pravilu
  - *List comprehensions*
- Moćan način za jednostavno kreiranje liste
  - Tipično u jednoj liniji koda
- Osnovna sintaksa:  
*lista = [izraz for iterator in sekvenca]*
  - **izraz** može biti sam **iterator**, poziv funkcije ili drugi izraz koji vraća vrednost
  - **iterator** je član ili vrednost iz sekvenice
  - **sekvenca** može biti lista, skup, generator ili bilo kakav objekat koji može da vrati svoje članove jedan po jedan

# Kreiranje liste prema pravilu (2)

---

- Primer – formiranje liste kvadrata brojeva
- Tradicionalan kod korišćenjem petlje:

```
def kvadrati_1(n):
    kvadrati = []
    for i in range (n+1):
        kvadrati.append(i*i)
    return kvadrati
```

- Elementi rešenja:
  - Kreiranje prazne liste
  - Iteriranje kroz sekvencu
  - Dodavanje elemenata na kraj liste

# Kreiranje liste prema pravilu (3)

---

- Kod napisan korišćenjem zadatog pravila

```
def kvadrati_2(n):  
    return [i*i for i in range(n+1)]
```

- Jednostavno kreiranje liste kroz zadati opis

- Poziv:

```
>>> kvadrati_1(10)  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  
>>> kvadrati_2(10)  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- Smatra se rešenjem koje je više u duhu programskog jezika Python ("Pythonic")
  - Često se koriste za kreiranje, mapiranje i filtriranje lista
  - Smatraju se jednostavnijim za čitanje i razumevanje

# Kreiranje liste prema pravilu (4)

---

- Filtriranje neželjenih vrednosti korišćenjem dodatnog uslovnog izraza:
  - `l = [izraz for iter in sekv if uslov]`
  - Uslovni izraz može biti bilo kakav validan izraz
- Primer – lista kvadrata brojeva deljivih sa tri

```
def kvadrati_3(n):  
    return [i*i for i in range(n+1) \  
           if i % 3 == 0]  
  
>>> kvadrati_3(10)  
[0, 9, 36, 81]
```

# Kreiranje liste prema pravilu (5)

---

- Umesti izraza koji definiše člana liste može da se stavi uslovni izraz:

- `l = [izraz1 if uslov else izraz2 \ for iter in sekv]`

- Primer – lista apsolutnih vrednosti

```
def aps_vred(arr):  
    return [i if i > 0 else -i for i in arr]  
  
">>>> aps_vred([3, -1, -5, 4, -17])  
[3, 1, 5, 4, 17]
```

- Uslovi mogu da se kombinuju

# Kreiranje liste prema pravilu (6)

---

- Primer – učitavanje liste celih brojeva sa standardnog ulaza po zadatom formatu
- Korišćenjem petlje:

```
def readNumbers():  
    niz = input().split(",")  
    for i in range(len(niz)):  
        niz[i] = int(niz[i])  
    return niz
```

- Korišćenjem pravila:

```
def readNumbers():  
    return [ int(i) for i in input().split(',') ]
```

# Kreiranje liste prema pravilu (7)

---

- Pravila se mogu ugnez̄davati
  - Pogodno kod kreiranja struktura koje liče na matrice
- Primer – formiranje matrice  $m \times n$

```
def matrica(m, n):  
    return [[i for i in range(n)] for j in range(m)]  
>>> matrica(3,4)  
[[  
    [0, 1, 2, 3],  
    [0, 1, 2, 3],  
    [0, 1, 2, 3]  
]
```

# Eratostenovo sito (1)

---

- Potrebno je formirati listu prostih brojeva od 2 do n
  - Poznati algoritam *The Sieve of Eratosthenes*
  - Počinje od broja 2 koji je najmanji prost broj
  - Iterativno pronalazi i obeležava umnoške prostih brojeva
    - Eliminišući ih tako iz daljeg razmatranja
- Postoji veći broj verzija algoritma
  - Neke su bolje optimizovane

# Eratostenovo sito (2)

- Primer rada jedne (optimizovane) varijante algoritma
  - Izvor slike: [https://en.wikipedia.org/wiki/Sieve\\_of\\_Eratosthenes](https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

# Eratostenovo sito (3)

---

- Rešenje korišćenjem petlji:

```
import math

def Eratosten_1(n):
    compounds = []
    for i in range(2, math.ceil(n**0.5) + 1):
        for j in range(i*2, n+1, i):
            compounds += [j]
    primes = []
    for i in range(2, n + 1):
        if not i in compounds:
            primes += [i]
    return primes
```

# Eratostenovo sito (4)

---

- Rešenje korišćenjem pravila:

```
def Eratosten_2(n):  
    compounds = \  
        [j for i in range(2, math.ceil(n**0.5) + 1) \  
         for j in range(i*2, n+1, i)]  
    primes = [x for x in range(2, n+1) \  
              if x not in compounds]  
  
    return primes  
  
>>> Eratosten_1(50)  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]  
>>> Eratosten_2(50)  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

# Kreiranje liste prema pravilu – ograničenja

---

- Upotrebljavati oprezno kod velikih lista
  - Liste se smeštaju direktno u memoriju

```
>>> sum([i * i for i in range(1000)])  
332833500
```
  - Bolje je koristiti generatorske izraze
    - Koristi se lenjo (*lazy*) izračunavanje

```
>>> sum(i * i for i in range(1000000))  
33333233333500000
```
- Slična pravila za kreiranje se mogu koristiti za skupove i rečnike
  - Biće obrađeno kasnije

Primetiti  
odsustvo []

Mogu se  
staviti ()



# Skupovi – tip `set` (1)

---

- Skup je kolekcija objekata nepromenljivih tipova
- Predstavlja neuređenu kolekciju objekata
- Ne pamti se redosled ubacivanja u skup
- Poredak elemenata pri ispisivanju ne mora odgovarati poretku pri dodeli ili formiranju

```
>>> boje = {'plava', 'crvena', 'bela', 'crna'}
```

```
>>> boje
```

```
{'crvena', 'bela', 'crna', 'plava'}
```

```
>>> 'bela' in boje
```

```
True
```

# Skupovi – tip `set` (2)

---

- Prazan skup se formira konstruktorom `set`:  
**A=set()**
  - Notacija `A={}` označava prazan rečnik, a ne skup!
- Može se formirati od sekvence (string, lista)
- Pri tome se duplikati izbacuju

```
>>> slova = set('abrakadabra')
>>> slova
{'k', 'r', 'b', 'd', 'a'}
>>> devojke=set(['Ana', 'Tanja', 'Maja', 'Ana'])
>>> devojke
{'Tanja', 'Maja', 'Ana'}
```

# Skupovi – tip `set` (3)

---

- Moguće je iterirati kroz skup:

```
>>>for n in boje:  
...     print(n)  
  
crvena  
bela  
crna  
plava  
  
>>>boje.add('zelena')  
  
>>>boje  
{'crvena', 'plava', 'zelena', 'bela', 'crna'}  
  
>>>boje.update(['a', 'b'])  
  
>>>boje  
{'crvena', 'a', 'plava', 'zelena', 'bela', 'crna', 'b'}
```

# Operatori za rad sa skupovima (1)

Operator	Metoda	Opis
$A \mid B$	<code>A.union(B)</code>	Unija skupova A i B
$A  = B$	<code>A.update(B)</code>	Dodaje sve elemente iz B u A
$A \& B$	<code>A.intersection(B)</code>	Presek skupova A i B
$A \&= B$	<code>A.intersection_update(B)</code>	U skupu A ostaje presek A i B
$A - B$	<code>A.difference(B)</code>	Elementi skupa A koji nisu u skupu B
$A -= B$	<code>A.difference_update(B)</code>	U skupu A ostaju oni koji nisu u B
$A ^ B$	<code>A.symmetric_difference(B)</code>	Elementi koji su samo u A ili samo u B
$A ^= B$	<code>A.symmetric_difference_update(B)</code>	Elementi koji su samo u A ili samo u B ostaju u skupu A

# Operatori za rad sa skupovima (2)

Operator	Metoda	Opis
<code>A == B</code>		Vraća <b>True</b> ako je skup A jednak B
<code>A != B</code>		Vraća <b>True</b> ako je skup A različit od B
<code>A &lt;= B</code>	<code>A.issubset(B)</code>	Vraća <b>True</b> ako je A podskup od B
<code>A &gt;= B</code>	<code>A.issuperset(B)</code>	Vraća <b>True</b> ako je A nadskup od B
<code>A &lt; B</code>		Vraća <b>True</b> ako je A pravi podskup od B
<code>A &gt; B</code>		Vraća <b>True</b> ako je A pravi nadskup od B
	<code>A.isdisjoint(B)</code>	Vraća <b>True</b> ako je presek A i B prazan

# Metode za rad sa skupovima

---

Metoda	Opis
<code>A.add(element)</code>	Ubacuje element u skup A
<code>A.discard(element)</code>	Uklanja element iz skupa A
<code>A.remove(element)</code>	Uklanja element iz skupa A (javlja grešku ako nije u skupu)
<code>A.clear()</code>	Uklanja sve elemente iz skupa
<code>A.pop()</code>	Vraća slučajno odabran element skupa i uklanja ga
<code>A.copy()</code>	Vraća kopiju skupa

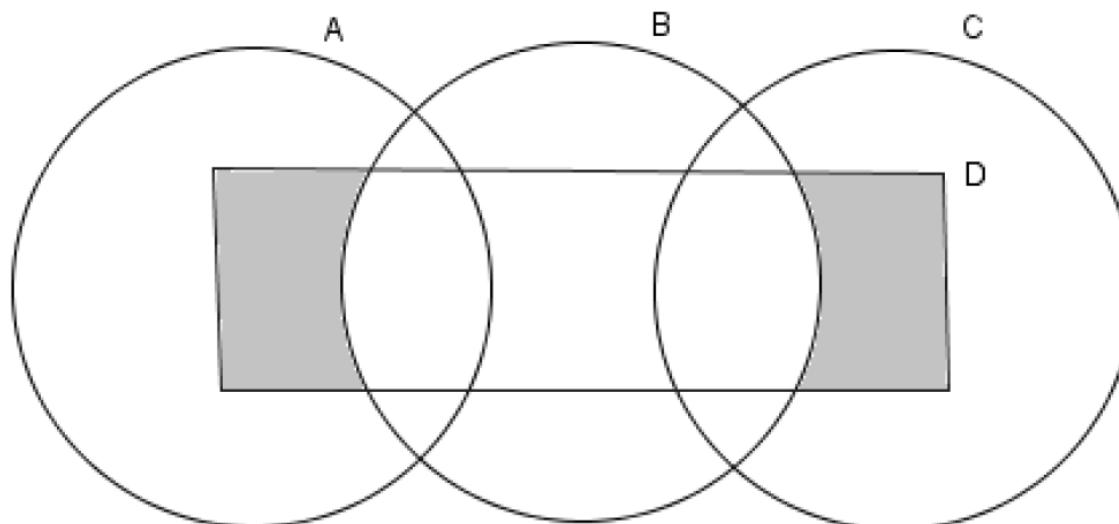
- Napomena:
  - Kada se uradi `A=B`, ubacivanje novog elementa u jedan od skupova, ubacuje ga i u drugi, a kod `A=B.copy()` ne!

# Skupovi – primer (1)

---

- Primer za 5. razred ☺
  - Izvor: <http://e.raspored.rs/mat/5/Skupovi.pdf>

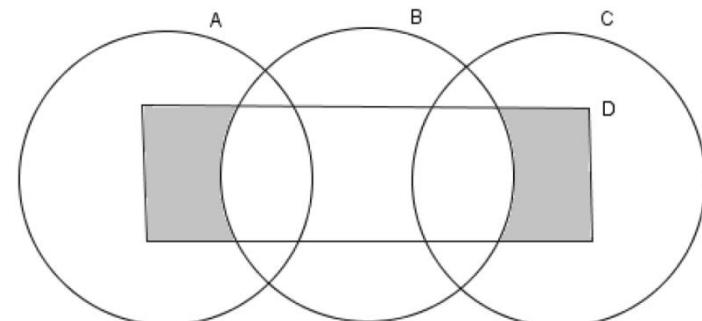
Na slici je Venov dijagram skupova  $A$ ,  $B$ ,  $C$  i  $D$ . Odrediti koji skup predstavlja osenčeni deo ovog dijagrama.



# Skupovi – primer (2)

---

- Uzmimo da su svi brojevi manji od 200:
  - U skupu A su neparni brojevi
  - U skupu C su parni brojevi
  - U skupu B su brojevi deljivi sa 3
  - U skupu D su brojevi deljivi sa 7
- $((A \cup B \cup C) \cap D) \setminus B$
- $((A \cap D) \cup (C \cap D)) \setminus B$
- $(D \setminus B) \cap (A \cup C)$



# Skupovi - primer (3)

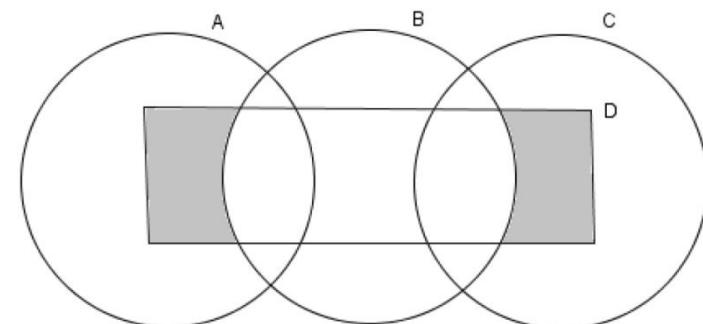
---

```
>>>A=set()
>>>B=set()
>>>C=set()
>>>D=set()
>>>for n in range (1,201):
...     if not n%2:
...         A.add(n)
...     else:
...         C.add(n)
...     if not n%3:
...         B.add(n)
...     if not n%7:
...         D.add(n)
>>>X=((A | B | C) & D) - B
>>>Y=((A & D) | (C & D)) - B
>>>Z=(D - B) & (A | C)
```

# Skupovi – primer (4)

```
>>>print(len(X), len(Y), len(Z))  
>>>if X==Y==Z:  
...     print ("Skupovi X, Y i Z su jednaki")  
>>>print ("Skupovi X, Y i Z sadrze: ", X)  
Skupovi X, Y i Z su jednaki  
Skupovi X, Y i Z sadrze: {133, 7, 140, 14, 154, 28,  
161, 35, 175, 49, 182, 56, 196, 70, 77, 91, 98, 112,  
119}
```

Parni i neparni (C i A),  
nisu deljivi sa 3 (B),  
jesu deljivi sa 7 (D)



# Skupovi - heterogenost

---

- U skupu se mogu nalaziti elementi različitih tipova, ukoliko su nepromenjivi

```
>>> svastara = { 'baba', 'žaba', 13, 'a', (1,2) }
>>> type(svastara)
<class 'set'>
>>> svastara
{(1, 2), 13, 'a', 'žaba', 'baba'}
```

- Dakle, lista ne može biti element skupa

```
>>> listaNe={'a', 1, [1,2]}
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    listaNe={'a', 1, [1,2]}
TypeError: unhashable type: 'list'
```

# Generisanje podskupova

---

- Python koristi metodu

`itertools.combinations(iterable, n)`

koja vraća podsekvence sekvence `iterable` dužine `n`

```
import itertools
def podskupovi(s, n):
    return [set(i) \
            for i in itertools.combinations(s, n)]
```

- Izvršavanje

```
s = {1, 2, 3, 4}
n = 3
print(podskupovi(s, n))
[{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}]
```

# Rečnici – tip `dict` (1)

---

- Rečnik (*dictionary*) je neuređena promenljiva kolekcija
  - Heš mapa ili heš tabela
- Služi za čuvanje parova objekata *ključ-vrednost*
  - Ključevi moraju biti *hashable* objekti
    - Odnosi se na većinu nepromenljivih objekata
- Omogućava dobijanje vrednosti na osnovu ključa
- Pri navođenju se ključ i vrednost u paru razdvajaju znakom :
- Parovi se navode u { } i razdvajaju zarezom
- Primer:

```
rečnik = { 'Ime' : 'Milan', 'Godine': 21, 'Smer': 'SI' }
```

# Rečnici – tip `dict` (2)

---

- Rečnik je neuređen
  - Pri ispisu redosled može biti drugačiji nego pri generisanju
- Prazan rečnik `A` može se dobiti na dva načina
  - `A = {}`
  - `A = dict()`
- Ubacivanje parova moguće eksplisitno, dodelom vrednosti `rečnik[ključ] = vrednost`

```
>>> recnik['Fak']='ETF'  
>>> recnik['Ime']='Ana'  
>>> recnik  
{'Ime': 'Ana', 'Godine': 21, 'Smer': 'SI', 'Fak': 'ETF' }
```

# Rečnici – nepostojeći ključ

---

- Ukoliko se referiše nepostojeći ključ biće prijavljena greška
  - Npr. `recnik['Prezime']`
  - Greška: `KeyError: 'Prezime'`
- Da bi se izbeglo generisanje greške, može da se koristi metoda `get()`

```
>>> ima_li = recnik.get('Prezime')
>>> print(ima_li)
None
```
- Može se metodi `get()` proslediti opcioni parametar koji vraća u slučaju nepostojećeg ključa

# Rečnici – nepostojeći ključ

---

```
>>> ima_li = recnik.get('Prezime', 'Nema' )  
>>> print(ima_li)  
Nema  
  
>>> ima_li == 'Nema'  
True  
  
>>> del recnik['Godine']  
>>> ima_li = recnik.get('Godine')  
>>> print(ima_li)  
None
```

# Metode za rad sa rečnicima (1)

Metoda	Opis
<code>A.clear()</code>	Uklanjanje svih elemenata iz rečnika
<code>A.copy()</code>	Vraća plitku ( <i>shallow</i> ) kopiju rečnika.
<code>dict.fromkeys(   seq[, v])</code>	Vraća novi rečnik sa ključevima iz <i>seq</i> i vrednostima <i>v</i> (ako vrednost nije data onda None)
<code>A.get(key[, d])</code>	Vraća vrednost za ključ <i>key</i> . Ako ne postoji , onda vraća <i>d</i> , a ako nije dato <i>d</i> vraća None
<code>A.items()</code>	Vraća <i>view</i> svih elemenata u rečniku u obliku ( <i>key, value</i> ).
<code>A.keys()</code>	Vraća <i>view</i> svih ključeva u rečniku
<code>A.pop(key[, d])</code>	Uklanja element sa ključem <i>key</i> i vraća njegovu vrednost, ili <i>d</i> ako nije pronađen, a ako je bez <i>d</i> javlja <i>KeyError</i> .
<code>A.popitem()</code>	Uklanja i vraća jedan proizvoljni element ( <i>key, value</i> ). Javlja <i>KeyError</i> za prazan rečnik

# Metode za rad sa rečnicima (2)

Metoda	Opis
<code>A.setdefault(key[, d])</code>	Ako je <i>key</i> u rečniku, vraća vrednost. Ako nije, dodaje <i>key</i> sa vrednošću <i>d</i> i vraća tu vrednost
<code>A.update([other])</code>	Ažurira rečnik parovima <i>key/value</i> iz <i>other</i> , ako neki ključ postoji, upisuje preko postojećeg
<code>A.values()</code>	Vraća <i>view</i> svih vrednosti u rečniku
<code>A.all()</code>	Vraća True ako su svi ključevi true, ili ako je rečnik prazan
<code>A.any()</code>	Vraća True ako je bilo koji ključ true. Za prazan vraća False.
<code>A.len()</code>	Vraća broj elemenata u rečniku
<code>A.sorted()</code>	Vraća novu,sortiranu listu ključeva u rečniku

# Rečnici – primer (1)

---

```
# prevodjenje binarnog stringa u oktalni
prevod = { '000':'0', '001':'1', '010':'2',
           '011':'3', '100':'4', '101':'5',
           '110':'6', '111':'7'}

def bin2oct(bin_cif):
    n = len(bin_cif)
    nule = bin_cif.count('0')
    jed = bin_cif.count('1')
    if nule + jed != n:
        return 'Greška'
```

## Rečnici – primer (2)

---

```
ost = n % 3
oct_cifre = []
if ost != 0:
    oct_cifre.append(prevod[ (3-ost)*'0' \
                           + bin_cif[0:ost]])
for i in range(ost, n, 3):
    oct_cifre.append(
        prevod[ bin_cif[i:i+3] ])
return ' '.join(oct_cifre)
```

# Torke – tip **tuple**

---

- Torka predstavlja uredjenu sekvencu
- Za razliku od liste, ne može da se menja po formiranju
- Može da sadrži elemente različitog tipa
- Često su elementi iz istog logičkog konteksta, odnosno odnose se na isti objekat iz realnog života
- Prilikom navođenja elemenata torke, oni se odvajaju zarezima
  - Navođenje može biti u zagradama () ili bez zagrada
- Ako torka sadrži samo jedan element, iza njega mora da стоји zarez
  - Npr. ('Pajton',)

# Torke - dodela

---

```
>>> nole = ('Novak', 'Đoković', 1987, 16 )
>>> type(nole)
<class 'tuple'>
>>> rafa = 'Rafael', 'Nadal', 1986, 19
>>> nole
('Novak', 'Đoković', 1987, 16 )
>>> rafa
('Rafael', 'Nadal', 1986, 19 )
>>> niko = ()
>>> niko
()
```

# Konstruktor `tuple()`

---

- Kao argument konstruktora `tuple()` navodi se validna sekvenca

```
>>> tuple('ETF')
```

```
('E', 'T', 'F')
```

```
>>> tuple([2000, 2011, 2019])
```

```
(2000, 2011, 2019)
```

```
>>> jedan = ('SI',)
```

```
>>> jedan
```

```
('SI',)
```

# Promena elemenata torke

---

- Pošto je torka nepromenljiv tip, pokušaj promene elementa torke izaziva grešku

```
>>> rafa[3] = 20
```

- Greška:
  - `TypeError: 'tuple' object does not support object assignment`
- Može se formirati nova torka koja sadrži neke elemente postojeće

```
>>> rafa_novo = rafa[0:3]+(20,)
```

```
>>> rafa_novo
```

```
('Rafael', 'Nadal', 1986, 20 )
```

# Torke – primer (2)

---

```
def analiza (tekst, rec):  
    x=tekst.count(rec)  
    y=tekst.find(rec)  
    z=tekst.rfind(rec)  
    return x,y,z  
  
tekst='ANA VOLI MILOVANA'  
rec='ANA'  
  
a,b,c = analiza(tekst, rec)  
print (a,b,c)  
# ispisuje: 2 0 14
```

# Literatura - knjige

---

- M. Kovačević, Osnove programiranja u Pajtonu, Akademска misao, Beograd, 2017.
- M. Lutz, Learning python: Powerful object-oriented programming, 5th edition, O'Reilly Media, Inc., 2013.
- J. Zelle, Python Programming: An Introduction to Computer Science, 3rd Ed., Franklin, Beedle & Associates, 2016.
- D. Beazley, B. K. Jones, Python Cookbook, 3rd edition, O'Reilly Media, 2013.
- A. Downey, J. Elkner, C. Meyers, How To Think Like A Computer Scientist: Learning With Python, free e-book

# Literatura – *online* izvori

---

- Python 3.8.0 documentation,  
<https://docs.python.org/3/index.html>
- Colin Morris, 7-day Python course,  
<https://www.kaggle.com/learn/python>
- Learn Python, Basic tutorial,  
<https://www.learnpython.org/>
- TutorialsPoint, Python tutorial  
<https://www.tutorialspoint.com/python/index.htm>