

Programiranje 1

Kontrolne strukture i liste

Univerzitet u Beogradu
Elektrotehnički fakultet
2019/2020.

Kontrolne strukture

Uvod u kontrolne strukture

- Koriste se za kontrolu toka programa
 - Obezbeđuju da određeni delovi programa budu izvršeni ako su ispunjeni određeni uslovi
- Dve vrste kontrolnih struktura
 - Uslovna grananja (selekcijske)
 - Omogućava se izbor jednog od većeg broja ishoda
 - Petlje (ciklusi)
 - Omogućavaju ponavljanje zadatih instrukcija
 - Određen broj puta (brojačke petlje)
 - Dok je određeni uslov ispunjen
- Uslovi se često definišu korišćenjem logičkih tipova

Logički tip podataka – tip `bool` (1)

- Uzima samo dve vrednosti:
 - Logičku istinu – `True`
 - Logičku neistinu – `False`
- Predstavlja rezultat logičkih i relacionih operatora
- Logičkom neistinom se u Python-u smatraju:
 - Vrednosti `False` i `None`
 - Vrednost 0 bilo kog numeričkog tipa: `0`, `0.0`, `0j`
 - Prazne sekvence i kolekcije
 - Postoji funkcija `bool()` za eksplisitnu konverziju tipa
- Sve druge vrednosti se smatraju logičkom istinom

Logički tip podataka – tip `bool` (2)

○ Logički operatori

Operator	Opis	Primeri
<code>and</code> - logičko I <code>x and y</code>	Ako je <code>x</code> neistinito, rezultat je <code>x</code> , u suprotnom <code>y</code>	$(a \text{ and } b) \rightarrow 20 \text{ (True)}$
<code>or</code> - logičko ILI <code>x or y</code>	Ako je <code>x</code> neistinito, rezultat je <code>y</code> , u suprotnom <code>x</code>	$(a \text{ or } b) \rightarrow 10 \text{ (True)}$
<code>not</code> – logička negacija <code>not x</code>	Negira logičko stanje svog operanda <code>x</code>	<code>not (a and b) → False</code>

U sledećim primerima, `a = 10`, `b = 20`.

Logički tip podataka – tip `bool` (3)

○ Relacioni operatori

Operator	Opis	Primeri
<code>==</code>	Ako su vrednosti oba operanda jednake, rezultat je tačan	$(a == b) \rightarrow \text{False}$
<code>!=</code>	Ako su vrednosti oba operanda jednake, rezultat je netačan	$(a != b) \rightarrow \text{True}$
<code>></code>	Ako je vrednost levog operanda veća od vrednosti desnog operanda, rezultat je tačan	$(a > b) \rightarrow \text{False}$
<code><</code>	Ako je vrednost levog operanda manja od vrednosti desnog operanda, rezultat je tačan	$(a < b) \rightarrow \text{True}$
<code>>=</code>	Ako je vrednost levog operanda veća ili jednaka od vrednosti desnog operanda, rezultat je tačan	$(a >= b) \rightarrow \text{False}$
<code><=</code>	Ako je vrednost levog operanda manja ili jednaka od vrednosti desnog operanda, rezultat je tačan	$(a <= b) \rightarrow \text{True}$

Uslovna grananja (1)

- Omogućavaju izvršavanje bloka koda pod uslovom da je neki logički uslov ispunjen
- Osnovna sintaksa:

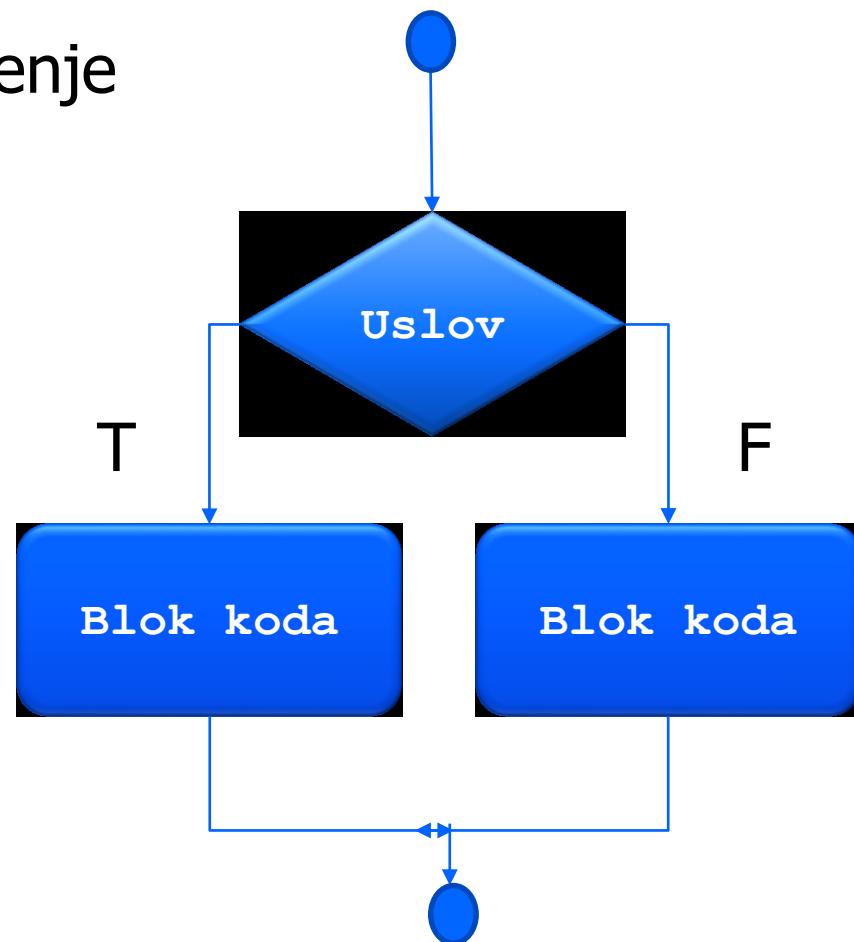
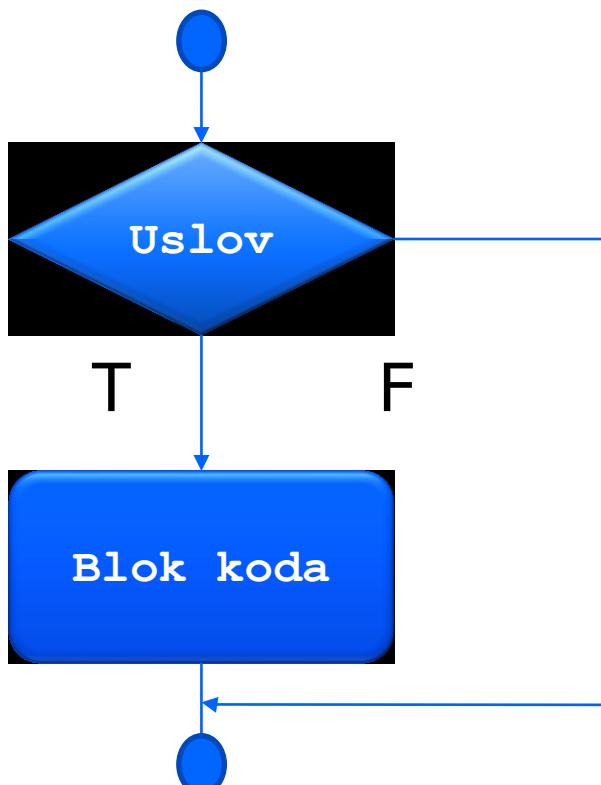
```
if uslov:  
    telo_then_grane # ako je uslov ispunjen  
else:  
    telo_else_grane # ako uslov nije ispunjen
```
- Izraz **uslov** mora biti izraz logičkog tipa
 - Mora se evaluirati u vrednost **True** ili **False**
 - Formira se korišćenjem relacionih i logičkih operatora, ili kao rezultat poziva funkcije
- Simbol dvotačka (:) zadaje početak novog bloka koda
 - Obavezno navođenje

Uslovna grananja (2)

- U zavisnosti od uslova biće izvršena samo jedna grana definisana blokom koda:
 - *telo_then_grane*, *telo_else_grane* su sekvenčne naredbi koje predstavljaju blokove koda
- Blok koda na Python-u sadrži proizvoljan broj naredbi
 - Najmanje jedna naredba
 - Ako se sastoji od samo jedne naredbe, ona se može navesti u produžetku (iza znaka :)
 - Ako ima više naredbi u bloku, svaka naredba mora biti u novom redu, podjednako uvučena u odnosu na **if-else** da bi se smatrala delom istog bloka

Uslovna grananja (3)

- Grana **else** se može izostaviti
 - Nije obavezno navođenje



Primer – rešenje linearne jednačine (1)

- Neka je data linearna jednačina $a*x+b = 0$
 - Ima rešenje ako $a \neq 0$, nema ako je $a == 0$
- Skica rešenja:

Šta uraditi?	Šta koristiti?
Učitati koeficijente jednačine: a, b	<code>input()</code>
Proveriti da li je $a \neq 0$	<code>if a != 0</code>
Ako jeste, odrediti i ispisati rešenje	<code>-b/a, print()</code>
Ako nije, proveriti da li je $b=0$, ili ne	<code>if b!=0: print("Jednacina nema resenja") else: print("Ima beskonacno mnogo!")</code>

Primer – rešenje linearne jednačine (2)

- Glavni deo rešenja

```
if a != 0:  
    x = -b / a  
    print("Resenje: ", x)  
else:  
    if b != 0:  
        print("Jednacina nema resenja")  
else:  
    print("Ima beskonacno mnogo!")
```

Primer – rešenje linearne jednačine (2)

Glavni deo rešenja

```
if a != 0:  
    x=-b/a  
    print("...")  
    print("Resenje: ", x)  
  
else:  
    if b != 0:  
        print("Jednacina nema  
        jednoznačno rešenje")  
    else:  
        print("Ima beskonacno  
        mnoogo ! ")  
        print("nema  
        rešenja")
```

Sve se smatra jednim blokom (else grana)

IF

Pod uslovom da je $a \neq 0$

F

print("Jednačina nema jednoznačno rešenje")

T

b != 0

F

)

print("nema
rešenja")

print("ima
beskonacno...")

Primer – rešenje linearne jednačine (3)

○ Kompletan kod

```
a = float(input("Uneti a: "))
b = float(input("Uneti b: "))

if a != 0:
    x = -b / a
    print ("Resenje:", x)
    #print ("Resenje: {:.2f}".format(x))
else:
    print("Nema jednoznacno resenje")
    if b != 0:
        print("Jednacina nema resenja")
    else:
        print("Jednacina ima beskonacno mnogo
resenja - svaki realan broj je resenje")
```

Obratiti pažnju na potrebu za konverzijom podataka sa ulaza

Višestruko uslovno grananje (1)

- Uslovno grananje **if-else** omogućuje međusobno isključivanje blokova koda
 - Izvršava se **then** blok u slučaju da je uslov ispunjen
 - Izvršava se **else** blok u slučaju da nije
- Međutim, često je potrebno ispitivanje višestrukih uslova
 - Uslovna grananja **if-else** se mogu ugnježđivati do proizvoljnog nivoa
- Može se koristiti **if-elif-else** varijanta
 - Omogućava proveru istinitosti više izraza
 - Izvršava se blok koda za prvi koji je istinit
 - Čak iako ima više istinitih
 - Slično kao **switch-case** višestruko grananje u drugim jezicima

Višestruko uslovno grananje (2)

- Primer – određivanje deljivost sa 2 i 3

- Ugnezđene **if-else** naredbe

```
num = int(input("Unesite broj: "))

if num % 2 == 0:
    if num % 3 == 0:
        print("Deljiv sa 2 i 3")
    else:
        print("Deljiv sa 2, nije deljiv sa 3")
else:
    if num % 3 == 0:
        print("Nije deljiv sa 2, deljiv sa 3")
    else:
        print("Nije deljiv sa 2, nije deljiv sa 3")
```

Višestruko uslovno grananje (3)

- Primer – određivanje ocene na osnovu broja poena
 - Korišćenje **if-else** naredbe

```
print("Odredjivanje ocene na Programiranju 1!")
poeni = float(input("Unesite broj poena:"))
if poeni > 90:
    ocena = 10
elif poeni > 80:
    ocena = 9
elif poeni > 70:
    ocena = 8
elif poeni > 60:
    ocena = 7
elif poeni > 50:
    ocena = 6
else:
    ocena = 5
print("Dobili ste ocenu:", ocena)
```

Uslovni izrazi (1)

- Python dozvoljava pisanje uslovnih izraza
 - Omogućavaju dodelu vrednosti promenljivoj u zavisnosti od nekog uslova
 - „Ternarni“ operator u drugim jezicima
 - Skraćeno pisanje uslovnih grananja
- Sintaksa:
 - **x if C else y**
 - U okviru izraza se najpre evaluira uslov **C**
 - Ako je **C** tačno, vrednost **x** se izračunava i vraća
 - Ako je **C** netačno, vrednost **y** se izračunava i vraća
 - Najmanji prioritet od svih operatora

Uslovni izrazi (2)

- Primer – ispis da li je student položio ispit

- Korišćenjem if-else kontrolne strukture

```
if ocena > 5:  
    poruka = "Da!"
```

```
else:  
    poruka = "Ne, pokusajte ponovo!"
```

```
print("Polozio:", poruka)
```

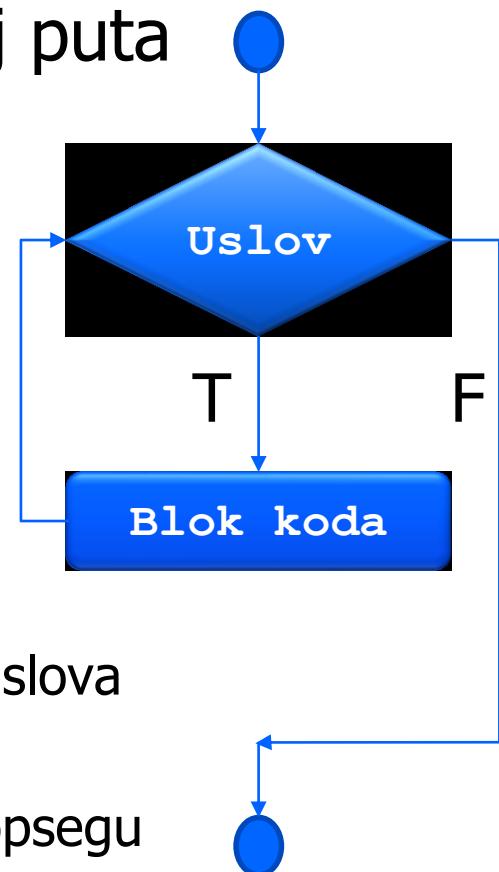
- Korišćenjem uslovnog izraza

```
poruka = "Da!" if ocena > 5 \  
           else "Ne, pokusajte ponovo!"  
print("Polozio:", poruka)
```

Da bi se naredba
protegla u više
redova dodaje se \

Petlje

- Petlje (ciklusi) se koriste kada god je neku obradu potrebno izvršiti veći broj puta
 - Do zadovoljenja određenog uslova
 - Određeni, zadati broj puta
 - Za svaki element kolekcije podataka
 - Za svaki znak u stringu
 - Za svaki element niza ili liste
- Python podržava dva tipa petlji:
 - **while**
 - Za iteriranje do zadovoljenja nekog opštег uslova
 - **for**
 - Za iteriranje kroz kolekcije ili u određenom opsegu
 - Esencijalno brojački ciklus



Petlja `while` (1)

- Ponavlja blok naredbi dok god je zadati logički uslov ispunjen
 - Provera uslova se radi pre izvršavanja tela petlje
 - Ciklus sa izlaskom na vrhu
 - Izvršava se 0 ili više puta

- Sintaksa:

`while uslov:`

`telo_petlje`

`#van petlje`

- Telo petlje se može sastojati od jedne ili više naredbi
 - Podjednako uvučenih u odnosu na ključnu reč `while`

Petlja while (2)

- Primer – Euklidov algoritam za određivanje NZD

```
m = int(input("Unesite m: "))
n = int(input("Unesite n: "))
m, n = (m, n) if m > n else (n, m)
a, b = m, n
r = a % b
while r > 0:
    a = b
    b = r
    r = a % b
nzd = b
print("NZD(", m, ", ", n, ")=", nzd)
```

Petlja `for` (1)

- Ponavlja blok naredbi određeni broj puta
 - U određenom, zadatom opsegu
 - Za svaki element neke kolekcije
- Nema logičkog uslova, već se petlji zadaje neki objekat koji omogućava iteriranje
- Sintaksa:

```
for iterator in sekvenca
    telo_petlje
```

 - Promenljiva petlje `iterator` redom uzima vrednosti iz sekvence
 - Za svaku vrednost iz sekvenice se obavlja jedna iteracija

Petlja `for` (2)

- Generisanje sekvence brojeva za iteriranje se vrši `range()` funkcijom

`range(stop)`

`range(start, stop, [step])`

- Generiše sekvencu u opsegu `[start, stop)`
- Ukoliko se ne zada start, podrazumeva se 0
- Donja granica se uključuje, gornja ne
- Polje `step` definiše korak koji mora biti različit od 0
 - Može biti negativan (omogućava iteriranje unazad)

- Primer:

```
>>> for broj in range(2, 7) : \
    print(broj, end=' ')
2 3 4 5 6
```



Podrazumevani kraj linije je novi red

2

3

4

5

6

Petlja `for` (3)

- Primer - ispisivanje sume svih parnih brojeva u zadatom opsegu od 1 do n

```
n = int(input("Unesi n: "));  
suma = 0
```

- Generisanje svih brojeva u opsegu uz test parnosti:

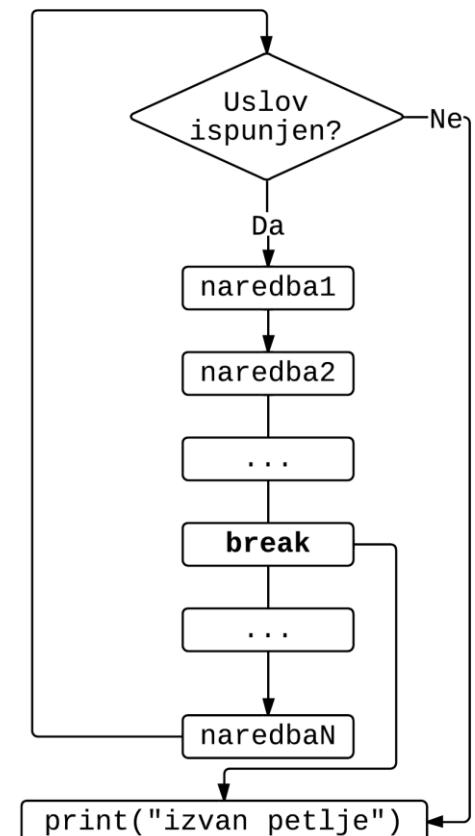
```
for i in range(1, n+1):  
    suma += i if i % 2 == 0 else 0  
print(suma)
```

- Generisanje sekvence samo parnih brojeva:

```
for i in range(2, n+1, 2):  
    suma += i  
print(suma)
```

Kontrola izvršavanja petlje (1)

- Povremeno postoji potreba da petlja ranije završi svoje izvršavanje
 - Može se koristiti naredba **break**
 - Napušta telo petlje na mestu na kome je izvršena
 - Nastavlja izvršavanje koda neposredno nakon tela petlje
 - Nasilan izlazak iz tela petlje zbog zadovoljavanja nekog eksternog uslova
 - Odnosi se na oba tipa petlje



Kontrola izvršavanja petlje (2)

- Naredba break omogućava:
 - Pravljenje beskonačnih petlji
 - Pravljenje petlji sa izlaskom na sredini ili dnu
- Primer – provera monotonosti sekvence

```
broj = -float("inf")
```

```
while True:
```

```
    unos = float(input("Unesite broj: "))
```

```
    if broj < unos:
```

```
        print("Sekvenca je rastuća!")
```

```
    else:
```

```
        print("Uneti broj prekida rastucu sekvencu!")
```

```
        break
```

```
    broj = unos
```

```
print("Kraj programa!")
```

Kod realnih brojeva
postoji konstanta
za beskonačnost

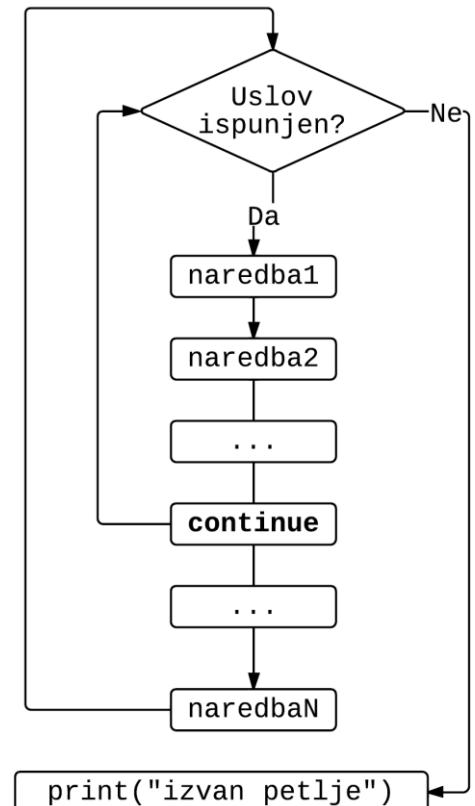
Petlja sa izlazom na dnu

- Postoji u nekim drugim jezicima (Pascal, C...)
- Obavezno se izvrši jednom
- Nakon izvršavanja tela petlje proverava se uslov izlaska ili ostanka u petlji

```
while True:  
    # telo petlje  
    # uslov nekad mora biti ispunjen  
    if uslov:  
        break  
    # naredbe posle petlje
```

Kontrola izvršavanja petlje (3)

- Povremeno postoji potreba da petlja prekine izvršavanje trenutne iteracije i pređe na sledeću
 - Može se koristiti naredba **continue**
 - Preskaču se sve naredbe do kraja bloka u okviru koga je pozvana
 - Nastavlja izvršavanje naredne iteracije petlje
 - Odnosi se na oba tipa petlje
 - Kod for petelje se vrši ažuriranje iteratora petlje



Kontrola izvršavanja petlje (4)

- Naredba **continue** se koristi u slučajevima kada u pojedinim iteracijama nije potrebno izvršiti sve naredbe petlje
 - Na primer za parne ili neparne brojeve
- Primer – suma nenegativnih parnih brojeva sa ulaza

```
suma = 0
while True:
    broj = int(input("Unesite broj: "))
    if broj < 0:
        break
    elif broj % 2:
        continue
    suma += broj
print("Suma = ", suma)
```

Za negativne brojeve napuštamo petlju

Za neparne brojeve preskačemo sumiranje

Kontrola izvršavanja petlje (5)

- Za **while** i **for** petlju postoji opcionalna **else** grana
 - Ukoliko je petlja regularno završena, onda se izvršava blok naredbi u **else** grani
 - Ne izvršava se ukoliko se petlja prekine sa **break**
- Primer – provera da li je broj prost

```
import math
```

Uključuje modul za rad
sa matematičkim funkcijama

```
n = int(input("Unesite broj: "))
for i in range(2, math.ceil(n**0.5) + 1):
    if n % i == 0:
        print("Broj nije prost!")
        break
else:
    print("Broj je prost!")
```

Prazan blok – **pass** naredba (1)

- U određenim situacijama je potrebno staviti praznu naredbu gde to sintaksno nije dozvoljeno
 - U grani **if**, **for** ili **while** naredbe, funkciji i sl.
 - Najčešće deo koda koji će biti implementiran kasnije
- Koristi se **pass** naredba
 - Izvršava se, ali se ništa ne dešava
- Primer – interaktivni meni

```
print("""  
    0. Unesi broj  
    1. Dodaj na sumu  
    2. Ispisi sumu  
    3. Pomoc  
    4. Izlazak  
""")
```

Prazan blok – pass naredba (2)

- Primer – interaktivni meni (nastavak)

```
broj = suma = 0
while True:
    izbor = int(input("Izbor: "))
    if izbor == 0:
        broj = int(input("Unesi broj: "))
    elif izbor == 1:
        suma += broj
    elif izbor == 2:
        print("Suma: ", suma)
    elif izbor == 3:
        # opcija će biti implementirana kasnije
        pass
    elif izbor == 4:
        break
    else:
        print("Nedozvoljen izbor!")
print("Kraj programa!")
```

Liste

Uvod u liste (1)

- Python poseduje ugrađeni tip liste
 - Složen tip koji predstavlja kolekciju drugih objekata
 - Lista predstavlja sekvencu objekata proizvoljnih tipova
 - Dozvljen je *type mixing* („babe i žabe“)
 - Lista može biti element liste
 - Predstavlja promenljiv objekat
 - Elementi se mogu menjati, dodavati i brisati
- Svakom elementu liste se dodeljuje indeks (pozicija) u listi na osnovu koje mu se može pristupiti
 - Indeksiranje se vrši od 0
- Postoji veliki broj ugrađenih funkcija za manipulaciju i obradu lista

Uvod u liste (2)

- Sintaksno se definiše pomoću [] zagrada

- Prazna lista

```
>>> emptyList = []
>>> print(emptyList)
[]
```

- Lista sa zadatim početnim sadržajem

```
>>> brojevi = [1, 2, 3, 4, 5]
>>> print(brojevi)
[1, 2, 3, 4, 5]
>>> pajton = ['python', 3, 3.8, True]
>>> print(pajton)
['python', 3, 3.8, True]
```

Pristup elementima liste (1)

- Pristup elementu liste:

```
>>> print(pajton[0] + str(pajton[1]))  
python3
```

Pretvara objekat u string

- Indeksiranje od 0, ali indeks može biti negativan

- Pozitivni indeksi predstavljaju pomeraje u odnosu na početak liste
- Negativni indeksi predstavljaju pomeraje u odnosu na kraj liste

- Element `a[-i]` predstavlja i -ti element s kraja liste `a`

```
>>> print(brojevi[-1])
```

5

Pristup elementima liste (2)

- Python omogućava pristup opsegu liste (*slicing*)
- Pravljenje podliste u opsegu elemenata [low, high)

```
>>> print("Python verzija: ", pajton[1:3])
Python verzija: [3, 3.8]
```

- Pravljenje podliste u opsegu [low, high) sa korakom step

```
>>> print("Neparni brojevi: ",
brojevi[0:5:2])
Neparni brojevi: [1, 3, 5]
```

Pristup elementima liste (3)

- Granice `low` i `high` se mogu izostaviti
 - Postavljaju se na vrednost početka ili kraja liste

```
>>> print("Brojevi: ", brojevi[2:])  
Brojevi: [3, 4, 5]
```

```
>>> print("Brojevi: ", brojevi[:3])  
Brojevi: [1, 2, 3]
```
- Granice `low` i `high` i korak `step` mogu biti negativni
 - ```
>>> print("Brojevi: ", brojevi[-2:])
Brojevi: [4, 5]
```

```
>>> print("Brojevi: ", brojevi[4:0:-2])
Brojevi: [5, 3]
```

# Ispitivanje pripadnosti kolekciji

---

- Python poseduje namenski operator **in** za ispitivanje pripadnosti elementa kolekciji
- Sintaksa:
  - **x in y**
    - Vraća **True** ukoliko **x** pripada kolekciji **y**, u suprotnom **False**
  - **x not in y**
    - Vraća **False** ukoliko **x** pripada kolekciji **y**, u suprotnom **True**
  - Ponašanje specifično za pojedinačne tipove
- Primer:

```
>>> print(5 in brojevi)
True
>>> print(15 in brojevi)
False
```

# Izmena elemenata liste

---

- Indeks elementa se može saznati funkcijom `index(object)`

```
>>> print(pajton.index(2))
1
```

- Izmena pojedinačnog elementa:

```
pajton[0] = 'python_old'
```

- Izmena grupe elemenata pristupom opsegu:

```
>>> pajton[1:3] = [2, 2.7]
>>> print(pajton)
['python_old', 2, 2.7, True]
```

# Iteriranje kroz elemente liste

---

- Tipično upotreboom for petlje

```
>>> for i in brojevi:
 print(i, end=' ')
1 2 3 4 5
```

- Obratiti pažnju da je iterator petlje **i** nepromenljiv

- Ukoliko je potrebno menjati članove liste u petlji, potrebno je dohvati ih po indeksu

```
>>> for i in range(len(brojevi)):
 brojevi[i] *= 10
>>> print(brojevi)
[10, 20, 30, 40, 50]
```

Funkcija određuje  
dužinu liste

# Dodavanje elemenata u listu (1)

---

- Inicijalizacija putem funkcije `list()` na osnovu objekta kroz koji se može iterirati

```
>>> etf = list("ETF")
>>> print(etf)
['E', 'T', 'F']
```

- Množenje liste brojem
  - Repliciranje elemenata liste određen broj puta
  - Korisno za inicijalizaciju kada je broj elemenata unapred poznat

```
>>> all_ones = [1] * 10
>>> print(all_ones)
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

# Dodavanje elemenata u listu (2)

---

- Dodavanje pojedinačnih elemenata na kraj liste
  - Funkcija `append()`:  
`>>> brojevi.append(6)`
- Dodavanje elementa na zadatu poziciju
  - Funkcija `insert(index, object)`
    - `pos` predstavlja poziciju umetanja
    - `object` predstavlja objekat koja se umeće
  - `>>> brojevi.insert(0, 10)`
  - `>>> print(brojevi)`
  - `[10, 1, 2, 3, 4, 5]`

# Dodavanje elemenata u listu (3)

---

- Dodavanje više elemenata na kraj liste:

- Korišćenjem operatora + za nadovezivanje

```
>>> brojevi += [7, 8, 9] # nakon append
>>> print(brojevi)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Korišćenjem funkcije `extend()`

```
>>> brojevi.extend([7, 8, 9])
```

- Funkcija `append` neće postići isti efekat:

```
>>> brojevi.append([7, 8, 9])
```

```
>>> print(brojevi)
```

```
[1, 2, 3, 4, 5, 6, [7, 8, 9]]
```

Lista kao element liste

# Uklanjanje elemenata iz liste (1)

---

- Uklanjanje pojedinačnog elementa ili opsega elemenata

```
>>> brojevi = [1, 2, 3, 4, 5, 6]
```

- Naredbom `del`

```
>>> del brojevi[3]
```

```
>>> print(brojevi)
```

```
[1, 2, 3, 5, 6]
```

```
>>> del brojevi[2:]
```

```
>>> print(brojevi)
```

```
[1, 2]
```

# Uklanjanje elemenata iz liste (2)

---

- Uklanjanje člana sa zadate pozicije uz dohvatanje člana

- Funkcijom `pop(index)`
  - Podrazumevana vrednost `index` je indeks poslednjeg elementa

```
>>> print(brojevi.pop())
6
>>> print(brojevi.pop(2))
3
>>> print(brojevi)
[1, 2, 4, 5]
```

- Uklanjanje člana po vrednosti

- Funkcijom `remove()`

```
>>> brojevi.remove(4)
>>> print(brojevi)
[1, 2, 5]
```

# Funkcije za rad sa listama (1)

---

- Primer:  

```
>>> brojevi = [10, 2, 3, 2, 6, 3]
```
- Funkcija **len**(*list*) vraća dužinu liste  

```
>>> print(len(brojevi))
6
```
- Funkcija **sum**(*list*) sabira elemente liste  

```
>>> print(sum(brojevi))
26
```
- Funkcija **min**(*list*) vraća najmanji element u listi
- Funkcija **max**(*list*) vraća najveći element u listi  

```
>>> print(min(brojevi), max(brojevi))
2 10
```
- Funkcija **reverse**() obrće poredak elemenata liste  

```
>>> brojevi.reverse()
>>> print(brojevi)
[3, 6, 2, 3, 2, 10]
```

# Funkcije za rad sa listama (2)

---

- Funkcija `sort()` vrši sortiranje liste
- Funkcija `sorted(list)` formira novu listu koja predstavlja sortiranu zadatu listu

```
>>> nova = sorted(brojevi)
>>> print(brojevi)
[10, 2, 3, 2, 6, 3]
>>> print(nova)
[2, 2, 3, 3, 6, 10]
>>> brojevi.sort()
>>> print(brojevi)
[2, 2, 3, 3, 6, 10]
```

- Funkcija `count(val)` prebrojavanje pojavljivanja određene vrednosti

```
>>> print(brojevi.count(3))
2
```

# Funkcije za rad sa listama (3)

- Primer – pronalaženje svih tročlanih podnizova koji sadrže minimalni element liste celih brojeva

```
n = int(input("Unesite broj elemenata liste: "))
lista = []
for i in range(n):
 lista.append(int(input()))
minimum = min(lista)
rezultat = []
for i in range(len(lista) - 2):
 trojka = lista[i:i+3]
 if minimum not in trojka:
 continue
 rezultat.append(trojka)
print(rezultat)
```

Unos:

2

3

1

5

4

Rezultat:

[[2, 3, 1], [3, 1, 5], [1, 5, 4]]

# Problem kopiranja liste

---

- Operator dodele vrednosti ne vrši kopiranje liste, već samo uvodi novi identifikator

```
>>> brojevi = [10, 2, 3, 2, 6, 3]
>>> brojevi2 = brojevi
>>> brojevi2[0] = 20
>>> print(brojevi)
[20, 2, 3, 2, 6, 3]
```

Problem !!!

- Problem se rešava nekom od tehnika kopiranja liste:

```
>>> brojevi2 = brojevi.copy()
>>> brojevi2 = list(brojevi)
>>> brojevi2 = brojevi[:]
```

- Problem ponovo može nastati u složenijim strukturama podataka, jer se prave plitke (*shallow*) kopije
  - Rešavanje tog problema je van opsega kursa

# 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>