Programski jezik Python - prezentacija 1

–Uvod. Kontrola toka. Osnovi rada sa numeričkim podacima –

Miloš Brajović

Elektrotehnički fakultet
Univerzitet Crne Gore

Osnovi računarstva II

Uvod

Miloš Brajović

Elektrotehnički fakultet
Univerzitet Crne Gore

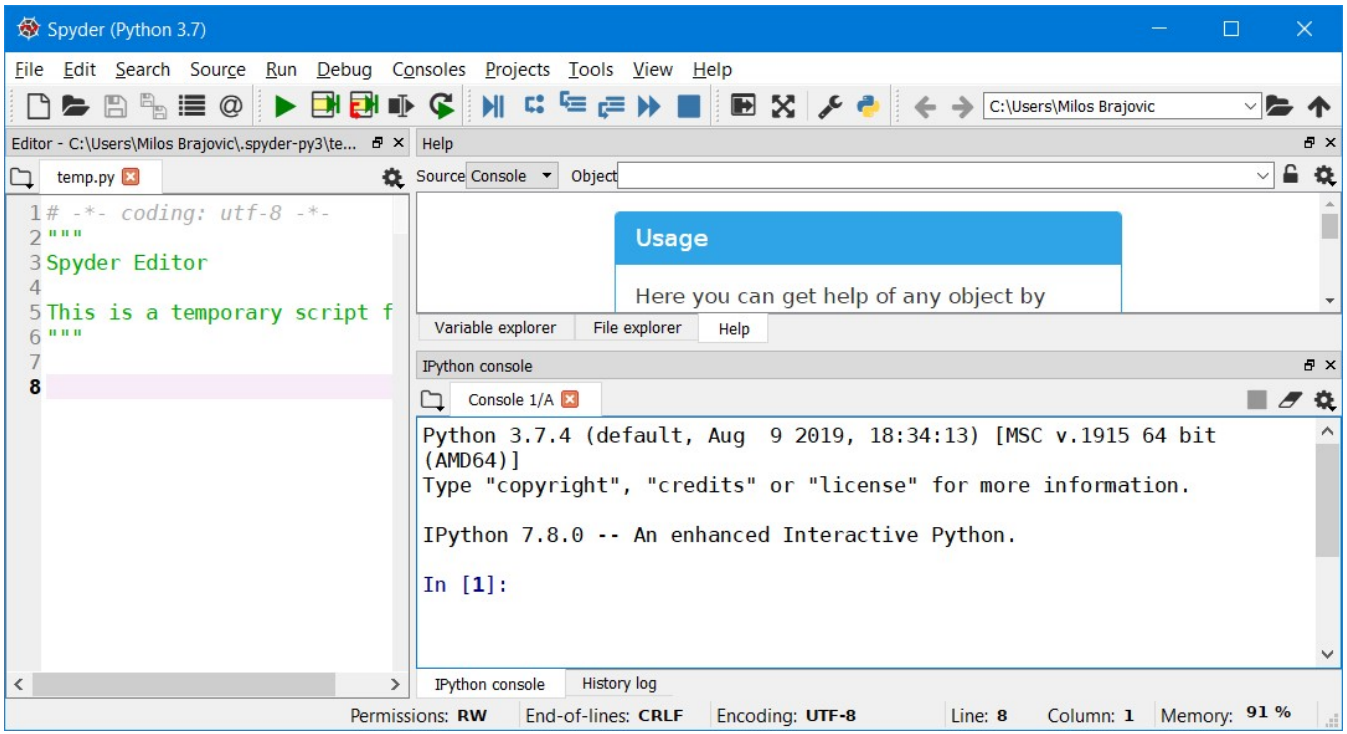
Osnovi računarstva II

Uvod

- Python je programski jezik opšte namjene, visokog nivoa.
- Instalacija Python-a se može preuzeti sa sajta https://www.python.org/.
- **IDLE** – je integrisano razvojno okruženje (Integrated Development Environment – IDE) koje je dio Python instalacije.
- Pomenimo još neka okruženja u kojima možete pisati Python programe:
  - **PyCharm**
  - **Spyder** – open source IDE optimizovan za data science (distribuira se često sa **Anaconda distribucijom**, koja se kategoriše kao Enterprise Data Science platform, prvobitno namijenjena naučnim proračunima, popularna u oblasti mašinskog učenja). Dobro se povezuje sa *SciPy*, *NumPy* i *Matplotlib*, koje su standardne ,,naučne" biblioteke.
  - **Thonny** – IDE namijenjen početnicima, itd.
- Za pisanje Python koda mogu se koristiti i druga okruženja: *Eclipse + PyDev*, *Sublime Text*, *Atom*, *GNU Emacs*, *Vi / Vim*, *Visual Studio*, *Visual Studio Code*
- Python programe možete pisati i u različitim web okruženjima, a jedno od njih je CodeScupltor3: https://py3.codeskulptor.org/

Spyder

- Spyder je dio Anaconda distribucije, koji je instaliran u računarskim salama ETF-a.
- U lijevom dijelu, naaslovljenom sa `temp.py` nalazi se editor u kojem je moguće sačuvati program. Dio naslovljen sa `Console 1/A` predstavlja komandnu liniju, u kojoj možemo sekvencijalno izvršavati naredbe.



Primjer programa

Potrebno je u meniju **File** odabrati opciju **New**, a zatim otkucati sljedeću sekvencu naredbi:

pozdrav.py

```
broj= eval(input('Unesite broj: '))
print('Kvadrat unijetog broja je: ', broj*broj)
```

- Posmatrani program treba sačuvati pod nazivom `pozdrav.py`.
- Obratiti pažnju na ekstenziju `.py` koju imaju Python programi.
- Za izvršavanje koristiti taster F5 ili opcije **Run module** u meniju **Run**.
- U komandnoj liniji/shell-u dobiće se rezultat:

```
Unesite broj: 4
Kvadrat unijetog broja je: 16
```

- Program je prvo ispisao tekst `Unesite masu u kg:` i čekao korisnika da unese broj; korisnik je unio broj 4, nakon čega se ispisao sadržaj druge linije gornjeg rezultata.

Primjer programa – komentari

- U prvoj liniji prethodnog programa definisano je šta će program ispisati, odnosno, tražiti od korisnika.
  - Funkcija `input` ispisuje tekst koji je naveden unutar zagrada pod navodnicima, kojim ona zapravo obavještava korisnika da nešto treba da unese, a zatim **preuzima to što je korisnik unio**.
  - Funkcija `eval` omogućava da se ono što je korisnik unio protumači kao numerička vrijednost.
  - U prvoj liniji koda navedena je i promjenljiva `broj` u koju se smješta numerička vrijednost koju je korisnik unio.
  - Obratiti pažnju da znak **=** označava operator **pridruživanja** sadržaja sa desne strane u promjenljivu koja je sa lijeve strane ovog operatora.
- U drugoj liniji vrši se računanje kvadrata broja i rezultat se ispisuje na ekranu.
  - Pomoću funkcije `print` vrši se prikaz, odnosno, *štampanje* rezultata na ekranu.
  - Dio unutar navodnika (odnosno, da budemo precizniji, apostrofa) predstavlja tekst (**string**) koji se ispisuje.
  - Zarez razdvaja prvi argument funkcije `print` od drugog argumenta, u kojem se promjenljiva `broj` množi (`*`) sa samom sobom.

Još jedan primjer

Zadatak: Napisati program koji od korisnika traži unost tri broja, a, b, c, koji predstavljaju stranice trougla, i koji računa poluobim trougla. Poznato je da je $O = (a+b+c)/2$.

Rješenje:

Poluobim_trougla.py

```
a=eval(input('Unesite prvu stranicu: '))
b=eval(input('Unesite drugu stranicu: '))
c=eval(input('Unesite trecu stranicu: '))
print('Poluobim trougla je: ',(a+b+c)/2)
```

- Prve tri linije koda omogućavaju da se preuzmu brojevi `a`, `b` i `c` koje je korisnik unio sa tastature u *shell*-u.
- Obratiti pažnju da su u `(a+b+c)/2` zagrade nephodne, jer dijeljenje (i množenje) ima veći **prioritet** u odnosu na sabiranje (i oduzimanje).

Oko unosa podataka

- **Razlikovanje malih i velikih slova**
  - Python je **case sensitive**, odnosno, razlikuje mala i velika slova. Stoga `Varijabla`, `varijabla` i `VARIJABLA` predstavljaju različite stvari!
- **Spaces (razmaci)**
  - Na početku linija koda, razmaci utiču na rezultat, dok na drugim mjestima ne utiču. Stoga sljedeći program neće raditi:

program_sa_greskom.py

```
masa = eval(input('Unesite masu u kg: '))
  print('U funtama to je',2.2046*masa)
```

program_koji_radi.py

```
print(  'Softversko inzenjerstvo'   )
print('Softversko inzenjerstvo'       )
```

Funkcije `input` i `print`

- Funkcija `input`
  - Osnovna struktura funkcije `input` je

```
<promjenljiva> = input(<poruka za korisnika>)
```

  - Funkcija `input` služi za preuzimanje stringova – tekstualnih podataka.

```
ime = input('Otkucajte kako se zovete: ')
print('Zdravo, ', ime)
```

  - U cilju preuzimanja numeričkih vrijednosti, koristi se funkcija `eval` koja vrši konverziju stringa u broj. Ako korisnik unese matematički izraz oblika `2^7`, funkcija `eval` će ga izračunati i pridružiti promjenljivoj sa lijeve strane.

```
N = eval(input('Unesite neki broj: '))
print('Vas broj na 7 je:', N^7)
```

- Funkcija `print`
  - Funkcijom `print` vrši se ispis na ekranu:

```
print('Programski jezik Python')
```

  - Uočimo da je neophodna upotreba zagrada.

Funkcija `print` – nastavak

- U prethodnom primjeru odštampao se tekst iz argumenta.
- Potrebno biti pažljiv u slučaju brojeva:

```
print('10+12')
print(10+12)
```

- Prethodne naredbe štampaju različite stvari: prva štampa `10+12` dok druga štampa `22`.
- Funkcija `print` može imati više argumenata. Razdvajaju se zarezom. Pri štampanju, funkcija automatski dodaje razmake između njih. Program:

stampanje.py

```
print('Ako se izracuna 5+9, dobija se', 14)
print('A', 22, 'XYZ', 81000)
```

daje

```
Ako se izracuna 5+9, dobija se 14
A 22 XYZ 81000
```

Funkcija `print` – nastavak

- Opcioni argument **`sep`**
  - Python dodaje razmake između argumenata funkcije **`print`**.
  - Opcioni argument **`sep`** može poslužiti da se umjesto razmaka definiše neko drugo slovo koje će se štampati između argumenata. Tako je moguće štampati `@` ako se definiše **`sep='@'`**, a moguće je štampati i više karaktera, npr. `##`, ako se zada **`sep='##'`**.
  - Moguće je spriječiti bilo kakvo umetanje, zadavanjem **`sep=''`**. Na primjer:

separator.py

```
print('Kvadrat od 21 je', 21*21, '.')
print('Kvadrat od 21 je ', 21*21, '.', sep='')
```

kao rezultat daje:

```
Kvadrat od 21 je 441 .
Kvadrat od 21 je 441.
```

- Opcioni argument **`end`**
  - Funkcija **`print`** nakon štampanja rezultata automatski prelazi u novi red.

Funkcija `print` – nastavak

- Opcioni argument `end`

```
print('Ovo je prvi red.')
print('Ovo je drugi red.')
```

biće prikazano kao:

```
Ovo je prvi red.
Ovo je drugi red.
```

  - Opcioni argument `end` može se koristiti za sprečavanje funkcije `print` da automatski pređe u novu liniju.
  - Na primjer:

stampanje2.py

```
print('Ovo je prvi red',end='@')
print('Ovo je drugi red')
```

  - Dobija se:

```
Ovo je prvi red@Ovo je drugi red
```

Promjenljive

- Šta radi sljedeći program?

temperature.py

```
t = eval(input('Unesite temp. u Celzijusima: '))
f_temp = 9/5*t+32
print('U Farenghajtima to je', f_temp)
if f_temp > 212:
  print('Temperatura je iznad tacke kljucanja.')
if f_temp < 32:
  print('Temperatura je ispod tacke zamrzavanja.')
```

- *Zadatak:* odrediti sadržaj promjenljivih `x`, `y` i `z` nakon izvršavanja koda:

```
x=3
y=4
z=x+y
z=z+1
x=y
y=5
```

Promjenljive – nastavak

- Sadržaj promjenljivih iz prethodnog zadatka se najlakše dobija ukoliko se kôd protumači liniju po liniju:
  1. U promjenljivu `x` se smješta vrijednost `3`.
  2. U promjenljivu `y` se smješta vrijednost `4`.
  3. U promjenljivu `z` se smješta vrijednost `x+y`, odnosno, vrijednost `7`.
  4. U promjenljivu `z` se smješta stara vrijednost iz `z` uvećana za `1`, odnosno, 8.
  5. U promjenljivu `x` se smješta sadržaj promjenljive `y`, odnosno, `4`.
  6. Promjenljiva `y` dobija vrijednost `5`.
- Da li bi bilo dozvoljeno koristiti `x=y=5`?
- Pravila oko imenovanja promjenljivih:
  - Nazivi varijabli mogu sadržati slova, brojeve i donju crtu (_).
  - Nazivi varijabli ne smiju sadržati razmake (space).
  - Nazivi varijabli ne smiju počinjati brojem.
  - Python je case sensitive pa treba voditi računa o malim i velikim slovima u nazivima varijabli.
  - Nazivi varijabli ne smiju biti ključne riječi Python-a.
- **Brisanje sadržaja varijabli**
  - `del ime_varijable` briše i varijablu
  - `ime_varijable=None` briše sadržaj varijable

Promjenljive – detaljnije, operatori, operandi

- Ključne riječi u Python-u su

```
and del from not while as elif global or with assert
else if pass yield break except import print class
in raise continue finally is return def for lambda try
```

- Pomoću funkcije `type` interpreter će nas obavijestiti o tipu varijable.
- Na primjer, `type('Pozdrav svima')` i `type('5.1')` u prvom slučaju javiće da je u pitanju **string** (`'str'`) a u drugom slučuaju **float** (`'float'`), odnosno, realan broj. Cio broj bi bio `'int'`.
- **Operatori** su specijalni simboli kojima se obavljaju osnovne operacije, dok su **operandi** vrijednosti nad kojima djeluju operatori.
  - `+, -, *, /, **` redom označavaju sabiranje, oduzimanje, množenje, dijeljenje, stepenovanje.
  - Operator `//` označava tzv. cjelobrojno dijeljenje.

Prioritet operacija

Od velike je važnosti zapamtiti kojim se redom izvršavaju operacije, odnosno, koje operacije imaju prioritet u odnosu na ostale. U programskom jeziku Python važe sljedeća pravila:

- Zagrade imaju najveći prioritet i pomoću njih se vrše sva eventualna razrješavanja prioriteta. Pošto se prvo izvršavaju izrazi u zagradama, za `2*(3-1)` će se dobiti `4`, dok `(1+1)**(5-2)` daje `8`.
- Stepenovanje je sljedeće po prioritetu. Tako `2**1+1` daje `3`, dok se za `3*1**3` dobija takođe `3`.
- Sledeći po prioritetu su množenje i dijeljenje (istog su prioriteta). Njihov prioritet je veći u odnosu na sabiranje i oduzimanje (kojima je isti prioritet). Stoga, `2*3-1` daje `5` dok se za `6+4/2` dobija `8`.
- Operatori istog prioriteta se izvršavaju sa **lijeva na desno**. Zato se u izrazu `5/3*2` prvo obavlja dijeljenje, pa tek onda množenje.

Operacije nad stringovima i komentari

- **Nije dozvoljeno**: `'2'-'1', 'jedan'/'dva', 'tri'*'ana'`
- **Konkatenacija**, odnosno, nadovezivanje, obavlja se operatorom **+**:

nadovezivanje.py

```
a='Prvi' #ovo je prvi string
b='drugi' #ovo je drugi string
print(a+b) #nadovezujemo i prikazujemo
```

- Ovdje se kao rezultat dobija `Prvidrugi`.
- Uočimo tekst iza simbola `#`. To su komentari, dio koda koji interpreter ignoriše. Komentari se mogu postaviti i na početak reda.
- **Ponavljanje** stringa je takođe zanimljiva operacija. Tako `print('Laptop'*3)` daje kao rezultat `LaptopLaptopLaptop`.

`for` petlja

Miloš Brajović

Elektrotehnički fakultet
Univerzitet Crne Gore

Osnovi računarstva II

`for` petlja

- `for` petlja omogućava ponavljanje sekvenci naredbi.
- Opšta sintaksa ove petlje u programskom jeziku Python je:

```
for naziv_varijable in range(broj_ponavljanja):
    naredbe koje treba ponavljati
```

  - Treba zapamtiti da se ključna riječ `for` mora napisati malim slovima.
  - Prva linija se mora završiti sa dvije tačke (`:`).
  - Naredbe koje se ponavljaju moraju na početku imati razmak.
- Sljedeći program će štampati tekst 'Dobar dan' 8 puta:

dobar_dan.py

```
for i in range(8):
    print('Dobar dan')
```

- Obratiti pažnju da pomoću razmaka Python zna šta treba ponavljati.

`for` petlja – detalji

- Varijabla `for` petlje
  - Posmatrajmo sljedeću petlju, koja štampa redom cijele brojeve od 0 do 99, svaki u posebnom redu:

stampaj_varijablu.py

```
for i in range(100):
    print(i)
```

  - Promjenljiva `i` se postavlja na **0** pri startovanju petlje.
  - Nakon svakog štampanja varijable `i`, program se vraća na početak petlje, i vrijednost `i` se uvećava za **1**.
  - Postupak se ponavlja 100 puta, a na kraju varijabla `i` ima vrijednost **99**.
  - Obratiti pažnju da je `i` samo naziv varijable. Umjesto njega, moglo je stajati, na primjer, `ime2`. Program bi dao isti rezultat.

`for` petlja – detalji

- Varijabla `for` petlje
  - Posmatrajmo još jedan primjer

stampaj_zdravo.py

```
for i in range(3):
    print(i+1, '-- Zdravo!')
```

```
1 -- Zdravo!
2 -- Zdravo!
3 -- Zdravo!
```

- Funkcija `range`
  - Funkcija `range` određuje koliko će puta biti ponovljena petlja. Proizvodi niz brojeva od **0** do broja u argumentu umanjenog za 1 (u gornjem primjeru, gornja granica je **2**).
  - Naravno, možemo zadati i proizvoljnu donju granicu. Tako, na primjer, `range(1,5)` generiše niz brojeva **1,2,3,4**. Da smo željeli brojeve od 1 do 5, pisali bi `range(1,6)`.

`for` petlja – funkcija `range`

- Brojevi koje generiše `range` ne moraju biti uzastopni cijeli brojevi. Trećim argumentom funcije možemo zadati neki drugi korak.
- Na primjer, `range(1,10,2)` daje brojeve od 1 do 9 sa korakom 2, odnosno, `1,3,5,7,9`.
- Brojevi mogu ići i unazad. Tako `range(5,1,-1)` daje brojeve od 5 do 2, unazad (korak -1), odnosno, `5,4,3,2`.
- Još primjera:

Naredba	Generisane vrednosti
`range(10)`	`0,1,2,3,4,5,6,7,8,9`
`range(1,10)`	`1,2,3,4,5,6,7,8,9`
`range(3,7)`	`3,4,5,6`
`range(2,15,3)`	`2,5,8,11,14`
`range(9,2,-1)`	`9,8,7,6,5,4,3`

`for` petlja – funkcija `range`

- *Zadatak.* Odštampati unazad brojeve od 5 do 1, u istom redu, kao i poruku **`Dobar dan!`**

odbrojavanje.py

```
for i in range(5,0,-1):
  print(i, end=' ')
print('Dobar dan!')
```

- *Zadatak.* Odštampati pravougaonik sastavljen od zvjezdica, koji ima 4 vrste i 6 kolona.

pravougaonik.py

```
for i in range(4):
  print('*'*6)
```

`for` petlja – funkcija `range`

- U prethodnom primjeru, konstrukcija `'*'*6` obezbjeđuje da ponavljanje stringa `'*'` (odnosno, konkretno, karaktera '*') 6 puta.
- *Zadatak.* Odštampati trougao, u čijem je prvom redu jedna zvjezdica, u drugom redu dvije, itd., do 5 zvjezdica.

trougao.py

```
for i in range(5):
  print('*'*(i+1))
```

- U prvoj iteraciji, varijabla `i` uzima vrijednost 0. Kako želimo da odštampamo u prvom redu jednu zvjezdicu, koristićemo `'*'*(i+1)`. U drugoj iteraciji, `i` uzima vrijednost 1, pa linija `'*'*(i+1)` štampa dvije zvjezdice (jer je `(i+1)`) sada 2, itd. Dakle, za vrijednosti varijable `(i)`, 0,1,2,3,4, imamo da `(i+1)`) uzima vrijednosti 1,2,3,4,5, pa se zato štampaju jedna, dvije, tri, četiri i pet zvjezdica.

Rad sa numeričkim podacima

Miloš Brajović

Elektrotehnički fakultet
Univerzitet Crne Gore

Osnovi računarstva II

Rad sa numeričkim podacima

- **Zapis brojeva**
  - U računarskoj tehnici, cijeli i decimalni brojevi se različito zapisuju.
  - Za decimalne brojeve koristi se *foating point* zapis.
  - Ovo znači da postoji ograničenje u preciznosti zapisa decimalnih brojeva.
  - Tipično je preciznost u Python-u 15 decimala. To znači da, na primjer, $\frac{7}{3} = 2.33333\ldots$, ne može biti reprezentovano sa beskonačnom preciznošću. Naime, ako u komandnoj liniji ukucamo `7/3`, dobijamo `2.3333333333333335`, odnosno, na petnaestoj decimali imamo *grešku u zaokruživanju*.
- **Osnovni operatori**

Operator	Značenje
`+`	sabiranje
`-`	oduzimanje
`*`	množenje
`/`	dijeljenje
`**`	stepenovanje
`//`	cjelobrojno dijeljenje
`%`	ostatak pri dijeljenju

Osnovni operatori

- **Cjelobrojno dijeljenje**
  - Obično dijeljenje, `/`, daje očekivani rezultat. Na primjer, `9/4` daje `2.25`
  - Cjelobrojno dijeljenje, `//` odbacuje decimalni dio, pa `9//4` daje `2`.
- **Ostatak pri dijeljenju**
  - Ostatak pri dijeljenju broja `a` sa `b` se zapisuje u obliku `a%b`.
  - Ako je `a%b` jednako `0`, tada je `a` djeljivo sa `b`.
  - Tako je `9%2` jednako 1, jer kada 9 podijelimo sa 4 dobićemo cjelobrojno 2, i ostatak 1.
  - Ako je broj `a` paran, važiće da je mu je ostatak pri dijeljenju sa 2, odnosno, `a%2` jednak 0.
  - Primjer upotrebe – kada želimo da u petlji neku operaciju obavljamo svaki drugi put (uz korišćenje IF-a, ispitujemo da li je ostatak pri dijeljenju brojačke varijable 0, i ako jeste – operaciju obavljamo, ako nije – preskačemo).
  - Zanimljiva je upotreba kada želimo da se vratimo na početak nekog ponavljajućeg ciklusa. Na primjer, imamo 12-časovni sat. Ako se izađe iz opsega od 12 sati, treba se vratiti na početak, npr. `(8+7)%12` daje 3h umjesto 15h.
- **Prioritet operacija – dodatak**: `*`,`/`, `//`, `%`, pa tek onda `+` i

Moduli. Slučajni brojevi

- **Moduli – Python biblioteke**
  - Osnovni elementi Python programskog jezika podrazumijevaju ključne elemente kao što su `for`, `if`, elementarni operatori, funkcije za unos i prikaz rezultata (`input`, `print`).
  - Ostali elementi dostupni su u tzv. **modulima** i moraju se uključiti, korišćenjem `import`-a, u sljedećem obliku:

```
from naziv_modula import sta_ukljucujemo
```

- **Slučajni cijeli brojevi – `randint`**
  - Funkcija `randint(a,b)` koja generiše slučajne cijele brojeve između a i b (uključujući a i b), dostupna je u modulu `random`, i može se uključiti na sljedeći način:

```
from random import randint
```

  - Primjer upotrebe:

```
from random import randint
x = randint(10,90)
print('Slucajan broj izmedju 10 i 90: ', x)
```

  - Svaki ponovni poziv `random` generiše novi slučajni broj između 10 i 90

Matematičke funkcije

- **Ugrađene funkcije**
  - Funkcije `abs` i `round` su ugrađene, odnosno, nije neophodno vršiti njihov *import*.
  - Fukcija `abs` računa apsolutnu vrijednost broja. Na primjer, `abs(-11.7)` kao rezultat daje `11.7`.
  - Funkcija `round` vrši zaokruživanje na najbliži cio broj, ili na najbližu decimalu. Zadaje se u obliku `round(a,n)`, gdje je prvi argument broj koji se zaokružuje, a drugi određuje na šta se zaokružuje prvi broj. Ako je $n > 0$, zaokruživanje se vrši n pozicija desno od decimalne tačke, za $n < 0$, zaokruživanje se vrši n pozicija lijevo od decimalne tačke, a za $n = 0$ imamo zaokruživanje na najbliži cio broj.
    - Na primjer, `round(5.8)` kao rezultat daje `6`.
    - `round(5.2)` kao rezultat daje `5`.
    - Sa druge strane, `round(128.5,-2)` kao rezultat daje `100.0`.
- **Modul `math`**
  - Osnovne matematičke funkcije dostupne su u okviru modula `math`.
  - Njihovo učitavanje se može obaviti naredbom `import math`.
  - Listu učitanih funkcija možemo dobiti komandom `dir(math)`. Šta radi funkcija `dir`?

Matematičke funkcije – modul `math`

- Pogledajmo spisak funkcija iz modula `math`:

```
>>> import math
>>> dir(math)
['__doc__','__loader__','__name__','__package__',
'__spec__','acos','acosh','asin','asinh',
'atan','atan2','atanh','ceil','copysign',
'cos','cosh','degrees','e','erf','erfc','exp',
'expm1','fabs','factorial','floor','fmod',
'frexp','fsum','gamma','gcd','hypot','inf',
'isclose','isfinite','isinf','isnan','ldexp',
'lgamma','log','log10','log1p','log2','modf',
'nan','pi','pow','radians','remainder','sin',
'sinh','sqrt','tan','tanh','tau','trunc']
```

- Sadržaj modula `math`
  - Uočiti da se pojavljuju nazivi fajlova koji počinju sa `__`
  - Obratiti pažnju na Euler-ov broj `e` i konstantu `pi`, odnosno, broj π.
  - Trigonometrijske funkcije, `sin`, `cos`, `tan` (argumenti su u radijanima)
  - Inverzne trig. funkcije, `asin`, `acos`, `atan`
  - Hiperboličke funkcije, `sinh`, `cosh`, `tanh`

Matematičke funkcije – modul `math`

- Sadržaj modula `math` – nastavak
  - Eksponencijalna funkcija `exp`
  - Logaritamske funkcije `log`, `log2`, `log10`,
  - Kvadratni korijen `sqrt`
  - Stepen, `pow(x,y,z=None)` , treći argument je opcion, `x**y%z`
  - Faktorijel `factorial`
  - Zaokruživanje `ceil`, `floor`
  - Specijalne konstante `inf`, `nan`
  - Najveći zajednički djelilac `gcd`
  - Funkcija za konverziju radijana u stepene `degrees`
  - Funkcija greške `erf` i inverzna funkcija greške, `erfc`
- Obratiti pažnju da se učitavanje modula može obaviti na više načina:

1. ```
   from math import sin, pi, exp
   # * umjesto naziva - ucitava se cio paket
   ```

   kada se funkcije/konstante koriste u obliku u kojem su učitane, `sin(pi/2)` daje `1.0`

2. ```
   import math
   ```

   kada se gornji primjer mora odraditi u obliku: `math.sin(math.pi/2)`

Još oko modula `math`

- Učitajmo modul `import math`
- Sadržaj modula: `help(math.sin)`
- Alternativno, ako učitamo sve funkcije, pomoću `from math import *`, tada pomoć tražimo u obliku `help (exp)`
- Ispitati sadržaj modula `random` i njegove funkcije
- Shell, odnosno, komandna linija kao kalkulator:

```
>>> 23**2
529
>>> s = 0 # inicijalizujmo varijablu s na 0
>>> for n in range(1,10001): #za n od 1 do 1000
s = s + 1/n**3 # sabirajmo
>>> s
1.202056898160098
>>> from math import * # ucitali smo sve iz modula mat
>>> factorial(13) % racunamo faktorijel
6227020800
```

- Računali smo sumu $s = \sum_{n=1}^{10000} \frac{1}{n^3} = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \ldots \frac{1}{10000^3} \approx 1.20205$

Selekcija

Miloš Brajović

Elektrotehnički fakultet
Univerzitet Crne Gore

Osnovi računarstva II

Selekcija – `if`

- Selekciju koristimo kada je potrebno da se određeni skup naredbi izvrši isključivo ukoliko je neki uslov ispunjen.
- *Zadatak:* Neka računar generiše slučajni broj iz opsega od 1 do 10 (koji ne ispisuje). Zatim od korisnika traži da pogodi koji je slučajni broj ,,uzeo". Ako je korisnik pogodio, računar treba da ispiše poruku o tome.

pogodi_broj.py

```
from random import randint
br=randint(1,10)
br_kor=eval(input('Unesite (pogodite) broj: '))
if br==br_kor:
  print('Pogodili ste broj!')
```

- Iza ključne riječi `if` stoji uslov, nakon kojeg stoji dvotačka (`:`). Ako je uslov tačan (True), izvršava se jedna ili više naredbi u narednim linijama, ispred kojih stoji razmak.

Selekcija – `if`

- *Zadatak:* U prethodnom primjeru, ništa se ne ispisuje ukoliko korisnik ne pogodi broj. Modifikovati primjer tako da se i u tom slučaju ispisuje prigodna poruka

pogodi_broj2.py

```
from random import randint

br=randint(1,10)
br_kor=eval(input('Unesite (pogodite) broj: '))
if br==br_kor:
  print('Pogodili ste broj!')
else: #inace, ako uslov iz if nije tacan...
   print('Nijeste pogodili broj! Broj je: ',br)
```

- Ključna riječ `else` omogućava da definišemo šta će se izvršiti u slučaju da uslov iz `if` dijela selekcije nije ispunjen (odnosno, nije tačan).

Selekcija – `if`: uslovni operatori

- Operatori poređenja su dati u tabeli ispod:

Operacija	Značenje
`a>b`	tačno ako je $a > b$
`a>=b`	tačno ako je $a \geq b$
`a<b`	tačno ako je $a < b$
`a<=b`	tačno ako je $a \leq b$
`a==b`	tačno ako je $a = b$
`a!=b`	tačno ako je $a \neq b$

- Logički operatori su:
  - `and`, označava logičku *i* operaciju
  - `or`, označava logičku *ili* operaciju
  - `not`, označava logičku negaciju

```
if poena>=50 and poena<60: # Ako su oba uslova tacna
  print('Ocjena je E.')
if skor>1000 or vrijeme>50: # Ako je bar jedan tacan
  print('Game over.')
if not (skor>1000 or vreme>50): #Ako nijedan nije tacan
  print('Game continues.')
```

Selekcija – `if`: – nastavak

- Što se prioriteta tiče, `and` ima veći prioritet u odnosu na `or`.
- Stoga, pisaćemo

```
if (skor<1000 or vrijeme>50) and broj_pokusaja==0:
  print('Game over.')
```

- **Česte greške su:** `if m=100`, `if x>1 or x<100:`, `if poena>=80 and <90:`
- Sa druge strane, u Python-u je dozvoljeno da se stavi `if 60<=poena<70:`

Klauzula `elif`

- Naredba `elif` može da se koristi u cilju izbjegavanja višestruke upotrebe `if`-a.
- Bitan aspekat je taj da se, ako je uslov u `elif` tačan, izvršava se islključivo dio naredbi vezan za taj `elif`, dok se ostali djelovi `if-elif-else` selekcije neće izvršavati.

Zadatak: Napisati program koji će, za zadati broj poena, ispisati ocjenu studenta. Ocjena 'A' se dobija ako je broj poena u opsegu od 90 do 100, ocjena 'B' u slučaju kad je broj poena od 80 od 90 itd. do ocjene 'F', koja se dobija za broj poena u opsegu od 0 do 49.

ocijeni.py

```
poena = eval(input('Unesite broj poena: '))
if poena>=90:
  print('A')
if poena>=80 and poena<90:
    print('B')
if poena>=70 and poena<80:
  print('C')
if poena>=60 and poena<70:
  print('D')
if poena>50 and poena<60:
  print('E')
else:
  print('F')
```

Selekcija – `if`: – nastavak

- *Zadatak:* Alternativno rješenje prethodnog zadatka:

ocijeni2.py

```
poena = eval(input('Unesite broj poena: '))
if poena>=90: #ako je broj poena >=90
  print('A')
elif poena>=80: #inace, ako je broj poena >=80...
  print('B')
elif poena>=70:
  print('C')
elif poena>=60:
  print('D')
elif poena>=50:
  print('E'):
else:
  print('F')
```

Primjer

- *Zadatak:* Napisati program koji broji koliko se kvadrata od 1^2 do 121^2 završava cifrom 4.

prebroj3.py

```
brojac=0

for i in range(1,122):
  if (i**2)%10==4:
    brojac=brojac+1
print('Takvih brojeva ima',brojac)
```

- Obratiti pažnju da `(i**2)%10` daje ostatak pri dijeljenju sa 10. Ako je cifra najmanje težine 4, tada je `(i**2)%10` jednako 4. U tom slučaju se vrši ažuriranje brojačke promjenljive, odnosno, prebrojavanje kvadrata i^2 koji su djeljivi sa 4.

Rad sa nizovima

- Nizovi u Python-u su specijalni slučaj složenijeg tipa podataka koji se naziva *lista*. Kroz nizove je moguće kretati se pomoću indeksa. Funkcija `len()` kao rezultat daje dužinu niza.
- Niz se može zadati u obliku: `X = [1, 6, 7, 9]`. Prvi element ima indeks 0, pa mu pristupamo kao `X[0]`.
- **Primjer.** Napisati program kojim se unosi niz brojeva, a koji zatim štampa koliko ima parnih elemenata u tom nizu.

```
X = eval(input('Unesite niz: '))
brojac = 0
for i in range(len(X)):
    if X[i]%2 == 0:
        brojac = brojac + 1

print('Broj parnih elemenata je:', brojac)
```

Rezultat:

```
Unesite niz: [1, 5, 6, 8]
Broj parnih elemenata je: 2
```

Rad sa nizovima

- Prethodni primjer je mogao biti odrađen i na sljedeći način:

```
X = eval(input('Unesite niz: '))
brojac = 0
for el in X:
    if el%2 == 0:
        brojac = brojac + 1

print('Broj parnih elemenata je:', brojac)
```

- Primijetimo da se na ovaj način može izbjeći upotreba indeksa.
- Rezultat:

```
Unesite niz: [1, 5, 6, 8]
Broj parnih elemenata je: 2
```

While petlja

Miloš Brajović

Elektrotehnički fakultet
Univerzitet Crne Gore

Osnovi računarstva II

While petlja

- **While** petlja se koristi kada želimo definisati skup koraka koje je potrebno ponavljati, ali kada broj ponavljanja nije poznat.
- Primjer je, recimo, program sa konverzijom temperature. Potrebno je, recimo, da program konvertuje temperature koje korisnik unosi, sve dok korisnik ne javi da je potrebno završiti pogram, na primjer, unošenjem neke nemoguće vrijendosti temperature:

```
t=0
while t!=-999:
  t=eval(input('Unesite temp. u C (za izlaz -999): '))
  print('U Farenhajtima to je: ',9/5*t+32)
```

- Očigledno, naredbe iz tijela **while** petlje se ponavljaju dok god je uslov iza ključne riječi **while** ispunjen.
- Budući da u ovom uslovu figuriše varijabla **t**, neophodno je prethodno izvršiti njenu inicijalizaciju (prva linija koda). U ovom slučaju, sama vrijednost na koju se inicijalizuje varijabla nije važna.
- Čim korisnik unese vrijednost -999, nakon njene konverzije u Farenhajte, više neće biti ispunjeno da je **t!=-999**, pa se time petlja prekida.

While petlja

- **While** i **for**
  - Jasno je da se **for** petlja može realizovati pomoću while petlje. Neka imamo **for** petlju koja računa zbir parnih brojeva između 1 i 100:

```
S = 0
for i in range(1,101):
  if i%2==0:
    S = S + i
print('Suma parnih brojeva izmedju 1 i 100 je: ',S)
```

  - Ekvivalentna **while** petlja je:

```
S = 0
i = 1
while i <= 100:
    if i%2==0:
        S = S + i
    i = i + 1 # primijetiti -- nije u if bloku
print('Suma parnih brojeva izmedju 1 i 100 je: ',S)
```

  - U slučaju **while** petlje, neophodno je prije otvaranja petlje inicijalizovati **i** na 1, a u samoj petlji, neophodno je uvećavati ovu varijablu za 1.

Beskonačna petlja

- Pomoću `while`, moguće je, iako najčešće greškom, definisati petlju koja se nikad ne prestaje izvršavati. To je beskonačna petlja. Na primjer

```
i = 0
while i<100:
  print(i)
```

- Jasno je da će uslov `i<100` biti uvijek ispunjen, jer se vrijednost varijable `i` ne mijenja tokom iteracija.
- Za prekid, najčešće je moguće koristiti kombinaciju CTRL + C na tastaturi. U drugim okruženjima je moguće pretlju prekinuti traženjem opcije *Restart Shell*.

Naredba `break`

- Pomoću naredbe `break` može se izvršiti prekidanje `for` ili `while` petlje prije njihovog završetka.

Literatura

- Brian Heinold, *A Practical Introduction to Python Programming*, 2012, [online] https://www.brianheinold.net/python/python_book.html
- Allen B. Downey, *Python for software design*, Cambridge University Press, 2009.
- https://www.w3schools.com/python/default.asp, poslednji put pristupano 5. marta 2020. godine