

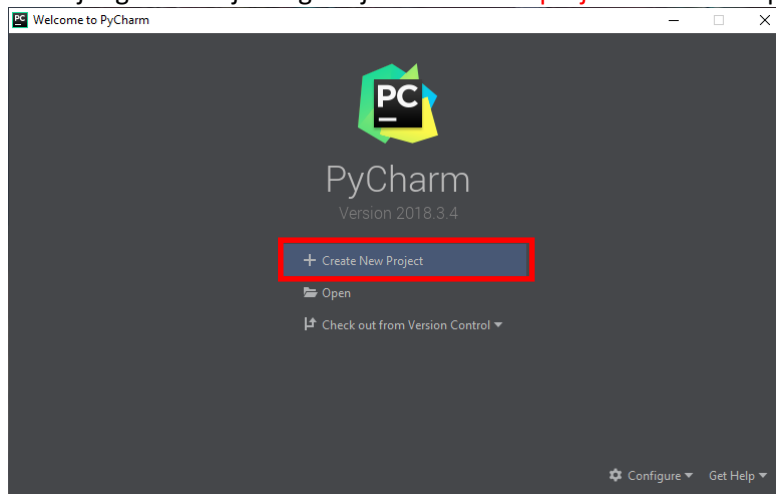
Python 3 podsetnik

Sadržaj

<i>PyCharm</i> projekat	2
<i>Python</i> program	3
Komentari	4
Promenljive	4
Tipovi vrednosti.....	4
Operatori.....	5
Izrazi	6
Literali	6
Naredbe	6
Blokovi naredbi	7
Selekcije	8
Petlje	9
<i>Try-except</i> blokovi.....	10
Funkcije	11
Biblioteke	14
Vidljivost promenljivih	15
<i>Python tuple</i>	16
Lista	16
Rečnik.....	17

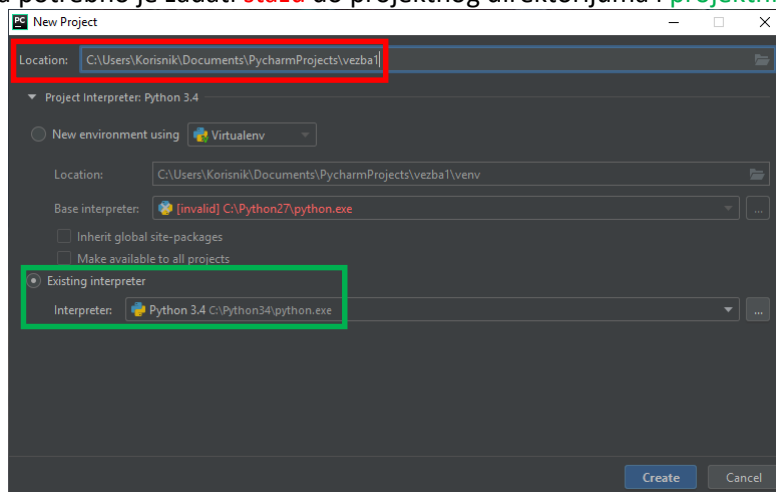
PyCharm projekat

Nakon pokretanja *PyCharm* razvojnog okruženja moguće je **kreirati novi projekat** ili otvoriti postojeći (slika 1).



Slika 1. *PyCharm* početni prozor

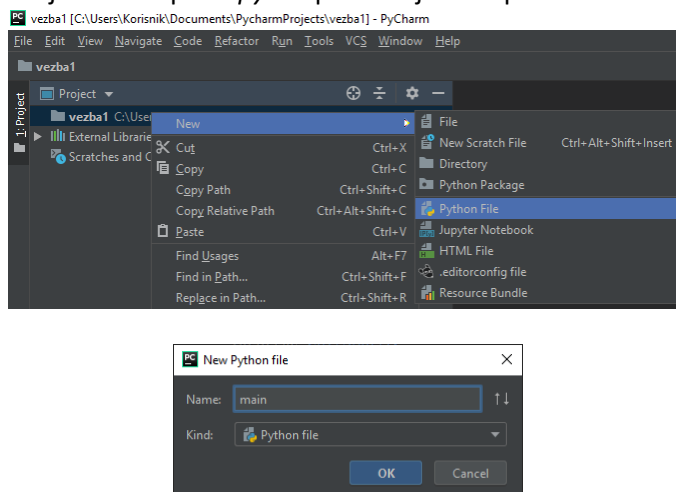
Pri kreiranju novog projekta potrebno je zadati **stazu** do projektnog direktorijuma i **projektni interpreter** (slika 2).



Slika 2. Kreiranje novog projekta

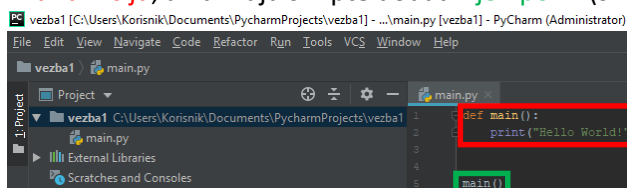
Python program

Nakon kreiranja projekta potrebno je kreirati prvu `.py` skriptu iz koje će započeti izvršavanje programa (slika 3).



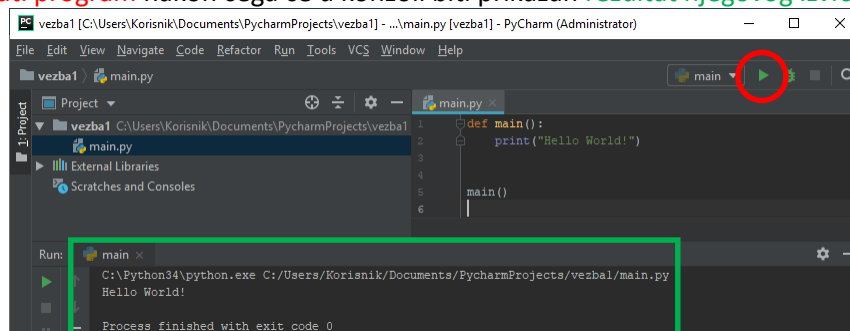
Slika 3. Kreiranje `.py` skripte

U skripti je potrebno definisati **glavnu funkciju**, a na kraju skripte dodati **njen poziv** (slika 4).



Slika 4. Glavna funkcija

Sada je moguće **pokrenuti program** nakon čega će u konzoli biti prikazan **rezultat njegovog izvršavanja** (slika 5):



Slika 5. Izvršavanje programa

Komentari

Komentari su elementi programa koji doprinose njegovoj čitljivosti, a koje interpreter ignoriše (slika 6). Pišu se za programera koji treba da čita kod.

```
"""
višelinijski
komentar
"""

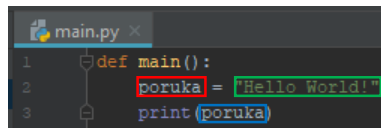
def main():
    # jednolinijski komentar
    print("Hello World!") # inline komentar

main()
```

Slika 6. Komentari

Promenljive

Promenljive su elementi programa koje služe da čuvaju vrednost pod nekim simboličkim nazivom. Pri definiciji, promenljivoj se moraju zadati **naziv** i **vrednost**. Nakon što je definisana, promenljiva se može **upotrebiti** proizvoljan broj puta u daljem izvršavanju programa (slika 7). Promenljiva se ne može deklarirati bez vrednosti. Promenljiva čuva vrednost sve dok se ona ne prepiše.



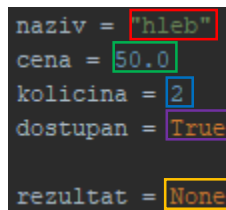
```
main.py x
1 def main():
2     poruka = "Hello World!"
3     print(poruka)
```

Slika 7. Promenljive

Tipovi vrednosti

Promenljive mogu da čuvaju vrednosti različitih tipova. 5 osnovnih tipova vrednosti u *Python* jeziku su (slika 8):

1. **string** (tekst)
2. **float** (realan broj)
3. **integer** (ceo broj)
4. **boolean** (logička vrednost; tačno/netačno)
5. **None** (prazna vrednost)



```
naziv = "hleb"
cena = 50.0
kolicina = 2
dostupan = True
rezultat = None
```

Slika 8. Osnovni tipovi vrednosti

Operatori

Operatori su elementi programa koji služe za izvođenje novih vrednosti na osnovu postojećih obavljanjem matematičkih i logičkih operacija nad njima. Način funkcionisanja, kao i tip rezultata operatora zavise od tipova vrednosti kojima rukuju (slika 9).

operatori

```
# aritmetički operatori
print(3 + 2) # sabiranje
print(3 - 2) # oduzimanje
print(3 * 2) # množenje
print(3 / 2) # deljenje
print(3 // 2) # celobrojno deljenje
print(3 % 2) # ostatak pri celobrojnem deljenju
print(3 ** 2) # stepenovanje

print()

# operatori poređenja
print(3 > 2) # veće
print(3 >= 3) # veće ili jednako
print(3 < 2) # manje
print(3 <= 3) # manje ili jednako
print(3 == 2, 3 == 3) # jednako
print(3 != 2, 3 != 3) # različito

print()

# string operatori
print("Hello" + " " + "World!") # konkatencija
print("Hello" * 2) # ponavljanje
print("Hello"[0], "Hello"[1], "Hello"[2], "Hello"[3], "Hello"[4]) # indeksiranje
print("Hello"[-1], "Hello"[-2], "Hello"[-3], "Hello"[-4], "Hello"[-5]) # indeksiranje od nazad
print("Hello"[1:4]) # indeksiranje u opsegu

print()

# logički operatori
print(True and True, True and False) # logičko "i"
print(True or False, False or False) # logičko "ili"
print(not True, not False) # negacija

print()

# pripadnost
print("H" in "Hello", "h" in "Hello")
print(0 in [1, 2, 3], 1 in [1, 2, 3])
```

rezultat

```
5
1
6
1.5
1
1
9

True
True
False
True
False True
True False

Hello World!
HelloHello
H e l l o
o l l e H
ell

True False
True False
False True

True False
False True
```

Slika 9. operatori

Provera da li neka promenljiva ima *None* vrednost se vrši operatorom *is* (Slika 9.1):

```
rezultat = izracunaj(podaci)
if rezultat is None:
    print("Došlo je do greške")
    return
```

Slika 9.1 Operator *is*

Izrazi

Izrazi su jednostavne ili složene konstrukcije u kojima mogu da figurišu operatori, varijable i pozivi funkcija, a čije izračunavanje rezultuje nekom vrednošću. Zagradama se mogu premostiti podrazumevani prioriteti operatora (slika 10).

izrazi

```
from math import sqrt

def main():
    ime = "Pera"
    a = 4

    print(3 + 2) # jednostavan izraz
    print("ime: " + ime) # jednostavan izraz
    print(3 + a) # izraz sa promenljivom
    print(3 + sqrt(9)) # izraz sa funkcijom
    print(3 + a + sqrt(a)) # složen izraz

    print()
    print(3 + 2 * 2) # množenje i deljenje imaju prednost u odnosu na sabiranje i oduzimanje
    print(3 + 2 * 2 ** 3) # stepenovanje ima prednost u odnosu na sabiranje, oduzimanje, množenje i deljenje
    print((3 + 2) * 2) ** 3 # zagrade premošćavaju podrazumevani prioritet operatora

main()
```

rezultat

```
5
ime: Pera
7
6.0
9.0

7
19
1000
```

Slika 10. Izrazi

Literali

Literali (za razliku od promenljivih) su nepromenljive vrednosti ugrađene u izraze (slika 11).

```
a = 3
b = a + 4
print(b + 3)

ime = input("Unesite ime: ")
print("ime: " + ime)
```

Slika 11. Literali

Naredbe

Naredba je osnovna jedinica izvršavanja programa. Program predstavlja sekvencu naredbi. **Izrazi** nemaju smisla ako nisu deo neke naredbe. Postoje 2 osnovna tipa naredbi (slika 12):

1. dodela vrednosti
2. poziv funkcije

```
a = 3 # dodela vrednosti
b = a + 4 # dodela vrednosti
print(b + 3) # poziv funkcije

ime = input("Unesite ime: ") # dodela vrednosti
print("ime: " + ime) # poziv funkcije

proizvodi = ucitaj() # dodela vrednosti
prikazi(proizvodi) # poziv funkcije
```

Slika 12. Naredbe

Blokovi naredbi

Blok naredbi je sekvenca naredbi koje se izvršavaju jedna za drugom. U *Python* jeziku blok naredbi je definisan uvlačenjem linija koda koje se navode iza znaka `“:”`. Sve uzastopne linije koda koje na svom početku imaju jednaku sekvencu *whitespace* karaktera (*tab*-ova ili razmaka) pripadaju istom bloku (slika 13). Sekvenca *whitespace* karaktera mora biti identična (ne mogu se mešati *tab*-ovi i razmaci za obeležavanje istog bloka).

```
def main():
    brojevi = []
    for it in range(5):
        broj = int(input("Unesite broj: "))
        brojevi.append(broj)
    print()
    print(brojevi)

main()
```

main funkcija nikada neće biti pozvana jer pripada bloku unutar svoje definicije

```
def main():
    brojevi = []
    for it in range(5):
        broj = int(input("Unesite broj: "))
        brojevi.append(broj)
    print()
    print(brojevi)
```

samo poslednji broj će biti dodat u listu jer se naredba nalazi u bloku unutar main funkcije, a ne bloku unutar for petlje

```
def main():
    brojevi = []
    for it in range(5):
        broj = int(input("Unesite broj: "))
        brojevi.append(broj)
    print()
    print(brojevi)
```

main()

Slika 13. Blokovi naredbi

Ako je potrebno, blok naredbi može biti prazan (slika 14).

```
def main():
    pass

main()
```

Slika 14. Prazan blok naredbi

Selekcije

Selekcije definišu blok naredbi koji se izvršava ako je prethodno zadovoljen uslov definisan logičkim izrazom (slika 15).

```
rezultat = izracunaj(podaci)
if rezultat is None: # uslov
    # ako je uslov zadovoljen
    print("Došlo je do greške")
    return
```

Slika 15. *If* blok

Selekcije mogu da uključe dodatni blok naredbi koji će biti izvršen ako uslov nije zadovoljen (slika 16).

```
broj = int(input("Unesite broj: "))
if broj > 0:
    # ako je uslov zadovoljen
    print("Broj", broj, "je prirodni broj!")
else:
    # u suprotnom
    print("Broj", broj, "je ceo broj!")
```

Slika 16. *Else* blok

Selekcije mogu i da uključe proizvoljan broj dodatnih uslova od kojih se svaki evaluira ako prethodni nije zadovoljen i za svaki od njih po jedan blok naredbi (slika 17).

```
bodovi = int(input("Unesite broj: "))
if bodovi >= 91:
    # ako je 1. uslov zadovoljen
    ocena = 10
elif bodovi >= 81:
    # ako je 2. uslov zadovoljen, a prethodni nije
    ocena = 9
elif bodovi >= 71:
    # ako je 3. uslov zadovoljen, a prethodni nisu
    ocena = 8
elif bodovi >= 61:
    # ako je 4. uslov zadovoljen, a prethodni nisu
    ocena = 7
elif bodovi >= 51:
    # ako je 5. uslov zadovoljen, a prethodni nisu
    ocena = 6
else:
    # ako prethodni uslovi nisu zadovoljeni
    ocena = 5

print()
print("Ocena:", ocena)
```

Slika 17. *Elif* blokovi

Petlje

While petlje definišu blok naredbi koji se ponavlja sve dok je prethodno zadovoljen uslov definisan logičkim izrazom (slika 18). One se koriste kada nije unapred poznato koliko puta neka sekvenca naredbi treba da se ponovi.

```
unos = ""
while unos == "": # uslov
    # ponavlja sve dok je uslov zadovoljen
    unos = input("Unesite tekst: ")
```

Slika 18. *While* petlja

For petlje definišu blok naredbi koji se ponavlja unapred poznati broj puta ili za svaki element unapred poznate kolekcije (slika 19).

```
brojevi = []
for it in range(5): # ponovi 5x
    broj = int(input("Unesite broj: "))
    brojevi.append(broj)

for it in [1, 2, 3, 4, 5]: # isto kao prethodno
    broj = int(input("Unesite broj: "))
    brojevi.append(broj)

suma = 0
for broj in brojevi: # ponovi za svaki element liste
    suma += broj

print()
print("Suma:", suma)
```

Slika 19. *For* petlja

Continue naredba preskače ostatak bloka naredbi. *Break* naredba preskače ostatak bloka naredbi i prekida petlju (slika 19.1).

```
print("Unesite maks. 5 naziva...")

nazivi = []
while len(nazivi) < 5:
    naziv = input("Unesite naziv (unesite 'x' za prekid): ")
    if naziv == "":
        continue
    if naziv == "x":
        break
    nazivi.append(naziv)

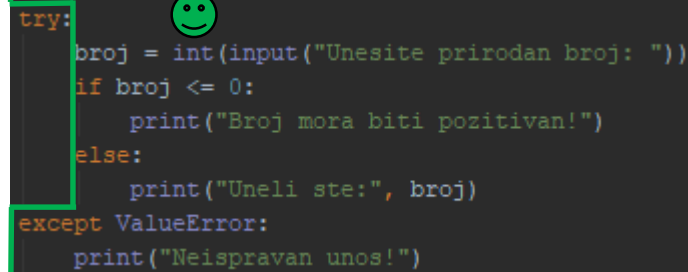
print()
print(nazivi)
```

Slika 19.1. *Continue* i *break* naredba

Try-except blokovi

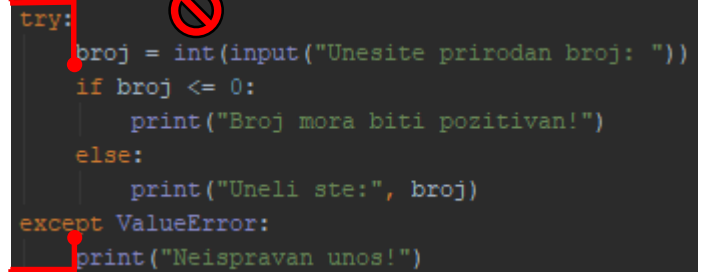
Mehanizam izuzetaka omogućuje da funkcije prijave da je u njihovom izvršavanju došlo do greške i da ne mogu da vrate nikakav smisleni rezultat. Tada se aktivira poseban tok kontrole, koji preskače sve ostale naredbe u *try* bloku i započinje se izvršavanje *except* bloka (slika 20).

tok bez izuzetka



```
try:
    broj = int(input("Unesite prirodan broj: "))
    if broj <= 0:
        print("Broj mora biti pozitivan!")
    else:
        print("Uneli ste:", broj)
except ValueError:
    print("Neispravan unos!")
```

tok sa izuzetkom



```
try:
    broj = int(input("Unesite prirodan broj: "))
    if broj <= 0:
        print("Broj mora biti pozitivan!")
    else:
        print("Uneli ste:", broj)
except ValueError:
    print("Neispravan unos!")
```

Slika 20. Try-except blok

Funkcije

```
def kalkulator():  
    a = None  
    while a is None:  
        try:  
            a = float(input("Unesite 1. operand: "))  
        except ValueError:  
            pass  
  
    operacija = ""  
    while operacija not in ["+", "-", "*", "/"]:  
        operacija = input("Unesite operaciju: ")  
  
    b = None  
    while b is None:  
        try:  
            b = float(input("Unesite 2. operand: "))  
        except ValueError:  
            pass  
  
    if operacija == '+':  
        rezultat = a + b  
    elif operacija == '-':  
        rezultat = a - b  
    elif operacija == '*':  
        rezultat = a * b  
    elif operacija == '/':  
        rezultat = a / b  
    else:  
        rezultat = None  
  
    print()  
    print("Rezultat je:", rezultat)  
  
kalkulator()
```

ponavljanje koda

Slika 21. Jednostavan program bez funkcija

Funkcije predstavljaju ponovno iskoristiv blok naredbi koji se može pozvati na više mesta u programu. Doprinosu i smanjenju kompleksnosti programa jer ga dele na manje logičke celine. Pre poziva, funkcija se mora definisati. Definicija funkcije sadrži naziv funkcije, skup parametara i blok naredbi (slika 21).

interpreter saznao za funkciju *unos_broja*

```
def unos_broja(poruka="Unesite broj: "): # podrazumevana vrednost parametra (ako se argument ne navede pri pozivu)
    broj = None
    while broj is None:
        try:
            broj = float(input(poruka))
        except ValueError:
            pass
    return broj
```

interpreter saznao za funkciju *unos_operacije*

```
def unos_operacije(): # bez parametara
    operacija = ""
    while operacija not in ["+", "-", "*", "/"]:
        operacija = input("Unesite operaciju: ")
    return operacija
```

interpreter saznao za funkciju *izracunaj*

```
def izracunaj(a, operacija, b): # sa više parametara
    if operacija == '+':
        rezultat = a + b
    elif operacija == '-':
        rezultat = a - b
    elif operacija == '*':
        rezultat = a * b
    elif operacija == '/':
        rezultat = a / b
    else:
        rezultat = None
    return rezultat
```

interpreter sada zna gde je funkcija *unos_broja*

interpreter saznao za funkciju *kalkulator*

```
def kalkulator():
    a = unos_broja("Unesite 1. operand: ")
    operacija = unos_operacije()
    b = unos_broja("Unesite 2. operand: ")

    rezultat = izracunaj(a, operacija, b)

    print()
    print("Rezultat je:", rezultat)
```

povratna vrednost
funkcije

argumenti funkcije

interpreter sada zna gde je funkcija *kalkulator*

kalkulator()

Slika 21. Funkcije

Postoji veliki broj korisnih ugrađenih funkcija (slika 22).

funkcije

```
broj_kao_tekst = input("Unesite broj: ") # čeka na unos teksta i vraća ga

print() # ispis praznog reda u konzoli
print(broj_kao_tekst) # prikazuje vrednost promenljive u konzoli
print("Uneli ste:", broj_kao_tekst) # prikazuje vrednost promenljive u konzoli sa razmacima

print()
ceo_broj = int(broj_kao_tekst) # konvertuje string u integer
print(ceo_broj)
realan_broj = float(broj_kao_tekst) # konvertuje string u float
print(realan_broj)

print()
print(round(2.3), round(2.7)) # zaokružuje na najbliži ceo broj
print(abs(-1)) # vraća apsolutnu vrednost broja
print(min(50, 80, 10), max(50, 80, 10)) # vraća min., odnosno maks. vrednost broja
print(min([50, 80, 10]), max([50, 80, 10])) # vraća min., odnosno maks. element liste

print()
brojevi = list(range(5)) # generiše listu celih brojeva
print(brojevi)
brojevi = list(range(2, 12, 2)) # generiše listu parnih brojeva
print(brojevi)

print()
duzina_liste = len(brojevi) # izračunava dužinu liste
print(duzina_liste)
duzina_stringa = len(broj_kao_tekst) # izračunava dužinu string-a
print(duzina_stringa)

print()
print(sorted([-5, 2, 8, -3, 1])) # sortira listu u prirodnom redosledu elemenata
print(sorted(["x", "c", "a", "b", "f"])) # sortira listu u prirodnom redosledu elemenata
```

rezultat

```
Unesite broj: 534

534
Uneli ste: 534

534
534.0

2 3
1
10 80
10 80

[0, 1, 2, 3, 4]
[2, 4, 6, 8, 10]

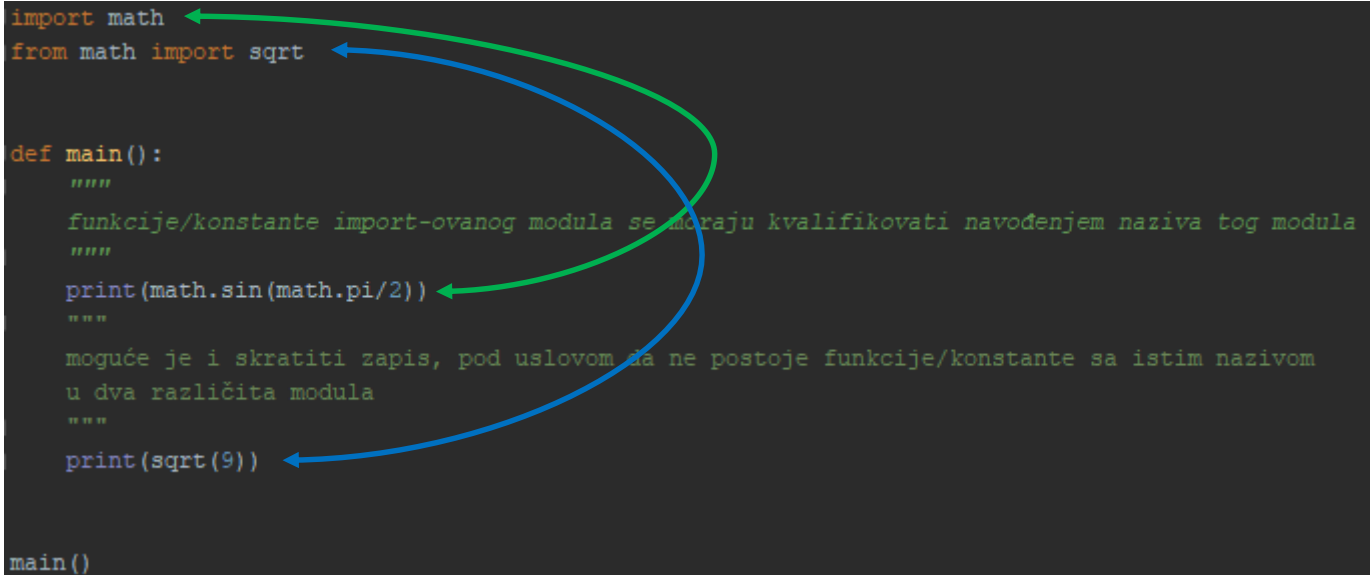
5
3

[-5, -3, 1, 2, 8]
['a', 'b', 'c', 'f', 'x']
```

Slika 22. Neke ugrađene funkcije

Biblioteke

Pored ugrađenih funkcija dostupne su i ugrađene biblioteke (slika 23), kao i korisnički definisane biblioteke (tzv. *3rd party* biblioteke). One se sastoje od jednog ili više *Python* modula (*.py* datoteke) okupljenih u svoj direktorijum (paket) i koji mora biti instaliran u direktorijumu interpretera ili kopiran neposredno uz *.py* skriptu iz koje je pokrenut program.



```
import math
from math import sqrt

def main():
    """
    funkcije/konstante import-ovanog modula se moraju kvalifikovati navođenjem naziva tog modula
    """
    print(math.sin(math.pi/2))
    """
    moguće je i skratiti zapis, pod uslovom da ne postoje funkcije/konstante sa istim nazivom
    u dva različita modula
    """
    print(sqrt(9))

main()
```

Slika 23. *Math* ugrađena biblioteka

Vidljivost promenljivih

Svaka promenljiva je vidljiva (je u *scope-u*) u okviru funkcije u kojoj je definisana. Promenljiva definisana na nivou modula je vidljiva u svim funkcijama tog modula (slika 24).

scope promenljive globalna_promenljiva

```
globalna_promenljiva = "globalna promenljiva"

def funkcija():
    scope promenljive promenljiva_u_funkciji
    promenljiva_u_funkciji = "promenljiva u funkciji"

    print(promenljiva_u_funkciji)
    print(promenljiva_u_main) # nije vidljivo u funkcija
    print(globalna_promenljiva)

def main():
    scope promenljive promenljiva_u_main
    promenljiva_u_main = "promenljiva u main"

    funkcija()

    print(promenljiva_u_funkciji) # nije vidljivo u main
    print(promenljiva_u_main)
    print(globalna_promenljiva)

main()

print(promenljiva_u_funkciji) # nije vidljivo na osnovnom nivou
print(promenljiva_u_main) # nije vidljivo na osnovnom nivou
print(globalna_promenljiva)
```

Slika 24. Vidljivost promenljivih

Python tuple

Python tuple je kolekcija koja grupiše više vrednosti. Nakon kreiranja, *tuple* je nepromenljiv. Kroz *tuple* se iterira u redosledu dodavanja elemenata. Vrednostima se pristupa po indeksu (slika 25). Vrednosti mogu biti različitog tipa.

upotreba

```
dani = (
    "ponedeljak",
    "utorak",
    "sreda",
    "četrvtak",
    "petak",
    "subota",
    "nedelja") # kreiranje

print(dani[0]) # pristup vrednostima

print()
for dan in dani: # iteracija kroz vrednosti
    print(dan)

print()
print("ponedeljak" in dani, "monday" in dani) # provera po sadržaju
```

rezultat

```
ponedeljak
ponedeljak
utorak
sreda
četrvtak
petak
subota
nedelja
True False
```

Slika 25. Python tuple

Lista

Lista je promenljiva kolekcija koja grupiše više vrednosti. Kroz listu se iterira u redosledu dodavanja elemenata. Vrednostima se pristupa po indeksu (slika 26). Vrednosti mogu biti različitog tipa.

upotreba

```
dani = [] # prazna lista
dani = [
    "utorak",
    "sreda",
    "četrvtak",
    "petak",
    "subota"] # popunjena lista

print(dani[0]) # pristup vrednostima

print()
for dan in dani: # iteracija kroz vrednosti
    print(dan)

print()
print("ponedeljak" in dani, "monday" in dani) # provera po sadržaju

dani.insert(0, "ponedeljak") # umetanje
dani.append("nedelja") # dodavanje na kraj
del dani[1] # brisanje

print()
print(dani)
```

rezultat

```
utorak
utorak
sreda
četrvtak
petak
subota
False False
['ponedeljak', 'sreda', 'četrvtak', 'petak', 'subota',
```

Slika 26. Lista

Rečnik

Rečnik je promenljiva kolekcija koja sadži parove: (ključ, vrednost). Kroz rečnik se iterira u nedefinisanim redosledu ključeva. Vrednosti se dodaju u rečnik pod ključem i pristupa im se po ključu pod kojim su dodati (slika 27). Ključevi i vrednosti mogu biti različitog tipa.

upotreba

```

proizvodi = {} # prazan rečnik
proizvodi = {
    "0001": "hleb",
    "0002": "mleko",
    "0003": "čokolada",
    "0004": "keks"} # popunjen rečnik

print(proizvodi["0001"]) # pristup vrednostima

print()
for sifra in proizvodi: # iteracija kroz ključeve
    print(proizvodi[sifra])
print()
for sifra in sorted(proizvodi): # iteracija prirodnim redosledu ključa
    print(proizvodi[sifra])

print()
print("0001" in proizvodi, "0000" in proizvodi) # provera po sadržaju

proizvodi["0005"] = "šećer" # dodavanje
del proizvodi["0001"] # brisanje

print()
print(proizvodi)

```

rezultat

```

hleb
hleb
keks
mleko
čokolada

hleb
mleko
čokolada
keks

True False
{'0004': 'keks', '0005': 'šećer', '0002': 'mleko', '0

```

Slika 26. Rečnik

Novi Sad, 2019.