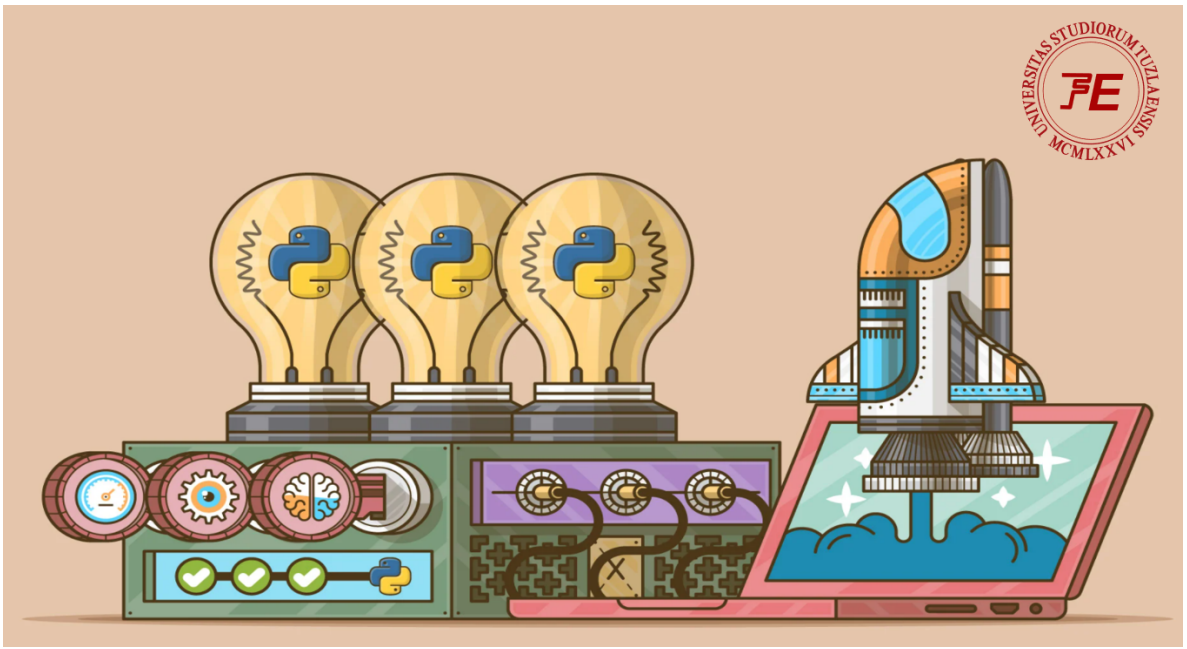
UNIVERZITET U TUZLI

FAKULTET ELEKTROTEHNIKE

Python tutoriali

Signali i sistemi

Emina Bajrektarević, Amera Sinanović



Tuzla,Septembar 2020. godine

Python Tutorijal 4

- **Definicija i poziv funkcije**
- **Parametri funkcije**
- **Naredba return**
- **Lokalne i globalne varijable**

Funkcija je dio programskoga koda koji je organiziran prema određenoj funkcionalnosti. Cilj korištenja funkcija jest da se dijelovi funkcionalnosti koji se ponavljaju u programskom kodu napišu samo jednom i to unutar funkcije. Korištenjem funkcija jedan te isti kod se može iskoristiti neograničen broj puta jednostavnim pozivom funkcije koja je u nekom trenutku potrebna. Ovakav pristup nam omogućava veću preglednost i organizaciju programskoga koda.

Funkcije se koriste kao osnovni elementi te se na temelju njih grade složenije strukture.Glavna misao vodilja prilikom kreiranja funkcija morala bi biti da funkcionalnosti funkcija budu što jednostavnije, tj. podijeljene na što manje logičke cjeline. Takva organizacija nam omogućuje veću ponovnu iskoristivost programskoga koda.

Funkcije mogu biti unaprijed napisane. Ako su funkcije unaprijed napisane, to su funkcije koje su ugrađene u sam programski jezik ili su dio standardnog ili uključenog paketa i o takvim funkcuija ćemo pričati u sklopu LV5.

Funkcija se sastoji od zaglavlja funkcije i tijela funkcije. Primjer definicije funkcije glasi:
def sumaBrojeva():
tijelo funkcije

Svaka funkcija koju želimo implementirati započinje ključnom riječju def nakon čega slijedi ime funkcije, u gornjem slučaju ime funkcije je sumaBrojeva(). Ime funkcije može biti proizvoljno, ali treba pripaziti da se funkcija ne zove poput neke od ključnih riječi u Pythonu ili imenom neke druge funkcije. Nakon imena funkcije obavezne su zagrade i nakon zagrada dvotačka. Ono što je bitno je to da tijelo funkcije mora biti uvučeno. Za razliku od drugih programskih jezika koji koriste vitičaste zagrade ili ključne riječi za razlikovanje programskih blokova, u Pythonu se koristi uvlačenje. U gornjem slučaju tijelo funkcije je programski blok koji je potrebno na neki način omeđiti kako bi se prilikom izvršavanja znalo koje sve linije programskoga koda ulaze u funkciju. Python koristi uvlačenje kao način razlikovanja programskih blokova. Povećanje uvlačenja znači da dolazi novi, ugniježđeni blok, smanjenje označava kraj trenutnog bloka programskoga koda.

Poziv funkcije

Nakon što je funkcija definisana i implementirana, možemo je pozvati tako da napišemo njeno ime te u malim zagradama navedemo odgovarajuće argumente. Sintaksa poziva funkcije je:
imeFunkcije()

```
def prviPoziv():
 print("Hello World!")
prviPoziv()
```

Kao što se može vidjeti iz gornjeg primjera, prvo je napisana funkcija (zaglavlje i tijelo funkcije) te je tek nakon toga izvršen poziv funkcije prviPoziv(). Ono što je bitno za primijetiti jest da definicija funkcije mora biti napisana ispred samog poziva funkcije. Ako se poziv funkcije u programskom kodu nalazi ispred definicije funkcije, kod pokretanja programa dogodit će se greška.

Parametri funkcije

Da bi funkcijama mogli slati različite vrijednost na temelju kojih će funkcije raditi neku obradu, u zaglavlju funkcije u malim zagradama upisuju se parametri koje funkcija prima.

Formalni parametri funkcije

Formalni parametri funkcije mogu se mijenjati po potrebi. U istom programu možemo nekoliko puta pozvati jednu te istu funkciju, recimo funkciju kvadrat(). Funkcija kvadrat() prima samo jedan parametar i zavisno od vrijednosti toga parametra ona izračunava kvadrat unesenoga broja. Prvi put pozovemo funkciju kvadrat(3), a drugi put kvadrat(5). Primijetimo da rezultat funkcije prilikom prvog poziva mora biti 9, a rezultat funkcije prilikom drugog poziva mora biti 25, tj. da rezultat zavisi od vrijednosti koje je funkcija primila. Formalni parametri su parametri, tj. varijable koje se nalaze u definiciji funkcije
def kvadrat(x):

Varijabla x je formalni parametar.

```
# -*- coding: utf-8 -*-
def kvadrat(x):
    print(x*x);

a=eval(input("Unesite broj: "))
kvadrat(a)
```

Iz priloženoga se vidi da funkcija prima jedan parametar naziva x. U tijelu funkcije na temelju vrijednosti u varijabli x izračunava se i ispisuje kvadrat prenesene vrijednosti. U pozivu prenosimo u funkciju vrijednost koju korisnik unese, Vrijednost se sprema u varijablu x. Prilikom poziva funkcije formalni parametri zamjenjuju se sa stvarnim argumentima, tj. s konkretnom vrijednošću.

Funkcije mogu primati i više parametara. Na primjer, ako funkcija mora vratiti sumu tri broja, a sve tri vrijednosti su funkciji predane kao parametri, i takva funkcija je prikazan u primjeru ispod.

```
# -*- coding: utf-8 -*-
# -*- coding: utf-8 -*-
def suma(x,y,z):
    print(x+y+z);

a=eval(input("Unesite prvi broj: "))
b=eval(input("Unesite drugi broj: "))
c=eval(input("Unesite treci broj: "))
suma(a,b,c)
```

Python nam omogućuje da unaprijed zadamo predefinirane vrijednosti parametara funkcije. To je omogućeno jer broj parametara funkcije ne mora uvijek odgovarati broju argumenata koji se šalju prilikom poziva funkcije. U nastavku je prikazan način pomoću kojega se postavlja parametru funkcije unaprijed zadana vrijednost. U donjem slučaju ta unaprijed zadana vrijednost je pridružena varijabli (formalnom parametru funkcije) var3 na vrijednost 0. Varijabla c će vrijednost 0 poprimiti samo ako se prilikom poziva funkcije prenesu 2 argumenta, a ne sva 3 argumenta koliko ih maksimalno može biti. Ako se prenesu 3 argumenta, tada varijabla c poprima vrijednost trećega prenesenog argumenta.

```
# -*- coding: utf-8 -*-
# -*- coding: utf-8 -*-
def suma(x,y,z=0):
    print(x+y+z);

a=eval(input("Unesite prvi broj: "))
b=eval(input("Unesite drugi broj: "))
c=eval(input("Unesite treci broj: "))
print("Poziv funkcije sa dva argumenta:")
suma(a,b)
print("Poziv funkcije sa tri argumenta:")
suma(a,b,c)
```

Naredba return

Do sada sve gore napisane funkcije u pozivajući dio koda nisu vraćale nikakvu vrijednost, već su samo ispisivale rezultat bilo kvadriranja, bilo sumiranja. Ako se u pozivajući dio koda želi vratiti vrijednost koju je funkcija, na primjer, izračunala, to se radi pomoću naredbe return.

```
# -*- coding: utf-8 -*-
def kvadrat(x):
    return x*x;

a=3
b=4
c=kvadrat(a);
d=kvadrat(b);
print(c+d)
```

U gornjem primjeru vidimo funkciju koja implementira matematičku operaciju kvadriranje. Za razliku od prethodnih funkcija, ova funkcija ne ispisuje rezultat kvadriranja. U ovoj funkciji rezultat kvadriranja zahvaljujući naredbi return postaje povratna vrijednost funkcije te se ta povratna vrijednost funkcije u gornjem primjeru kod prvog poziva funkcije kvadriranje() sprema u varijablu c, a kod drugog poziva funkcije povratna vrijednost se sprema u varijablu d.

Vidljivost i životni vijek varijable

Vidljivost varijable je pojam koji određuje iz kojih dijelova programa je neka varijabla dohvatljiva, tj. dostupna. Varijable mogu biti lokalne ili globalne. Životni vijek varijable je pojam koji je određen trenutkom u kojem je varijabla nastala u memoriji i trenutkom u kojem se ta ista varijabla briše iz memorije. Životni vijek varijable ograničen je završetkom bloka programskoga koda u kojem je ta varijabla nastala, na primjer, funkcija.

Lokalne varijable su varijable koje su korištene unutar tijela funkcija, u ovu kategoriju možemo svrstati i parametre funkcija. Načelno možemo smatrati da se parametri funkcija ponašaju identično kao i obične varijable unutar tijela funkcije.Lokalne varijable vidljive su samo unutar funkcije u kojoj su prvi put upotrijebljene. Prvom upotrebom može se smatrati izraz pridruživanja vrijednosti varijabli, na primjer, var=10. Ako je varijabla definirana unutar funkcije, nakon što se izvođenje funkcije završi (bilo pomoću naredbe return ili pak završetkom tijela funkcije) varijabla se trajno uništava. Što znači da sljedeći poziv iste funkcije više ne može vidjeti vrijednost varijable iz prethodnog poziva (jer je varijabla prilikom izlaska iz funkcije trajno uništena). Primjer upotrebe lokalnih varijabli:

```
def test():
```

```
 var=5

test()
print(var)
```

U gornjem programu u glavnom dijelu programa pozvana je funkcija test(), unutar funkcije test() definirana je varijabla var s vrijednošću 5. Nakon što izađemo iz funkcije (funkcija ništa ne ispisuje već samo postavlja vrijednost varijable var) u glavnom programu pokušavamo ispisati vrijednost varijable var, no varijabla var nakon izlaska iz funkcije više ne postoji. Nakon izlaska iz funkcije životni vijek varijable je završio tako da te varijable više nema ni u memoriji, a k tome varijabla var je lokalna varijabla te joj nije moguće pristupiti iz glavnog dijela programa jer je vidljiva samo unutar funkcije.

Uz lokalne postoje i globalne varijable. Globalne varijable su varijable koje su kreirane i korištene u glavnom dijelu programa i njihov životni vijek je tako dug dok se program izvršava. Globalne varijable se zovu globalne jer osim što se njima može pristupati iz glavnog dijela programa, tim varijablama se može pristupati i iz funkcija.

```
def test():
 print(var)

var=5
test()
```

U gornjem programu kreirana je varijabla var i pridružena joj je vrijednost 5. Nakon toga u sljedećoj liniji je pozvana funkcija test(). Funkcija test() ispisuje sadržaj varijable var, no primijetimo da funkcija nema ni formalnih parametara ni varijabli koje bi bile definirane u njoj, no da svejedno uspijeva ispisati vrijednost varijable var. To znači da je varijabla var globalna varijabla koja je vidljiva iz svih dijelova programa (što uključuje i funkcije).

Zadatak 4.1

Napišite funkciju koja prima 4 parametara. Funkcija mora ispisati rezultat matematičke formule: $((a * a) + (b * c) - d)/2$ U glavnom dijelu programa pozovite napisanu funkciju.

Rješenje

```
# -*- coding: utf-8 -*-
"""
Napi ite funkciju koja prima 4 parametara. Funkcija mora ispisati
rezultat matemati ke formule: $( (a*a) + (b*c) - d ) / 2$
U glavnom dijelu programa pozovite napisanu funkciju.
"""

def moja_funkcija(a,b,c,d):
    return ((a*a + b*c)-d)/2

a=5;
b=2;
c=3;
d=4;

y=moja_funkcija(a, b, c, d);
print(y)
```

Zadatak 4.2

Napišite funkciju mnozenje() koja može primiti 2 vrijednosti, od kojih je druga vrijednost unaprijed zadana, sami odredite broj koji ćete pridružiti unaprijed zadanoj vrijednosti. Funkcija mora izračunati i ispisati rezultat množenja. U glavnom dijelu programa pozovite funkciju dva puta. Kod prvog poziva funkcije, kao argumente funkcije navedite dvije vrijednosti (vrijednost drugog argumenta neka bude drugačija od unaprijed zadane vrijednosti). Drugi put funkciju pozovite samo jednim argumentom.

Rješenje

```
# -*- coding: utf-8 -*-
"""
Napi ite funkciju mnozenje() koja mo e primiti 2 vrijednosti, od
kojih je druga vrijednost unaprijed zadana, sami odredite broj koji
 ete  pridru iti unaprijed zadanoj vrijednosti. Funkcija mora izra unati
i ispisati rezultat mno enja.
U glavnom dijelu programa pozovite funkciju dva puta. Kod prvog
poziva funkcije, kao argumente funkcije navedite dvije vrijednosti
(vrijednost drugog argumenta neka bude druga ija od unaprijed
zadane vrijednosti). Drugi put funkciju pozovite samo jednim
argumentom.
"""

def mnozenje( var1, var2 =1):
    return var1*var2;

a=mnozenje(2);
b=mnozenje(2,3)
print(a)
print(b)
```

Zadatak 4.3

Napišite funkciju koja prima 2 parametara. Rezultat izračuna funkcije se ne ispisuje u funkciji nego u glavnom dijelu programa. Funkcija mora izračunati rezultat formule: $(a*a)+(b*b)$ Rezultat spremite u varijablu koja se nalazi u dijelu programskoga koda u kojem se funkcija poziva te ispišite tu varijablu.

Rješenje

```
# -*- coding: utf-8 -*-
"""
Napi ite funkciju koja prima 2 parametara. Rezultat izra una funkcije
se ne ispisuje  u funkciji nego u glavnom dijelu
programa. Funkcija mora izra unati rezultat formule:
$(a*a) + (b*b)$
Rezultat spremite u varijablu koja se nalazi u dijelu programskoga
koda u kojem se funkcija poziva te ispi ite tu varijablu.
"""

def moja_funkcija(a,b):
    return a*a + b*b

var1=2;
var2=5;

y= moja_funkcija(var1,var2);
print(y)
```

Zadatak 4.4

Potrebno je kreirati 4 funkcije koje će omogućiti sabiranje, oduzimanje, množenje i dijeljenje dva broja. Unose brojeva omogućite unutar funkcija. Unutar funkcije se mora izvršiti određena operacija i ispisati rezultat iste. Izbornik izraditi pomoću while petlje.

Rješenje

```
"""
 Potrebno je kreirati 4 funkcije koje  e  omogu iti sabiranje, oduzimanje, mno enje
    i
 dijeljenje dva broja. Unose brojeva omogu ite unutar funkcija. Unutar funkcije
 se mora izvr iti odre ena operacija i ispisati rezultat iste. Izbornik izraditi
 pomo u while petlje.
"""

def saberi():
    a=eval(input("Unesite prvi broj:"))
    b=eval(input("Unesite drugi broj:"))
    print(a+b)

def oduzmi():
    a=eval(input("Unesite prvi broj:"))
    b=eval(input("Unesite drugi broj:"))
    print(a-b)

def pomnozi():
    a=eval(input("Unesite prvi broj:"))
    b=eval(input("Unesite drugi broj:"))
    print(a*b)

def podijeli():
    a=eval(input("Unesite prvi broj:"))
    b=eval(input("Unesite drugi broj:"))
    print(a/b)

while 1:
    print("Dobro do li u kalkulator.py")
    print("Izaberi ra unsku operaciju: ")
    print("****************************** ")
    print("1-sabiranje")
    print("2-oduzimanje")
    print("3-mno enje")
    print("4-dijeljenje")
    print("5- Izlaz iz kalkulatora")
    print("****************************** ")
    operacija=eval(input(""));

    if operacija==1:
        saberi()
    elif operacija==2:
        oduzmi()
    elif operacija==3:
        pomnozi()
    elif operacija==4:
        podijeli()
    elif operacija==5:
        break
```