Programski jezik Python

Šantić, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studiji informatike

PROGRAMSKI JEZIK PYTHON

Završni rad

Tomislav Šantić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 27.02.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Tomislav Šantić
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	A4159, 22.09.2011.
OIB studenta:	99512557867
Mentor:	Doc.dr.sc. Damir Blažević
Sumentor:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ivan Aleksi
Član Povjerenstva:	Dr.sc. Ivan Vidović
Naslov završnog rada:	Programski jezik Python
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Opisati povjesni razvoj, značajke i primjenu programskog jezika Python. Opisati sintaksu, dostupna razvojna okruženja i mogućnosti. Izraditi nekoliko aplikacija i prikazati njihov razvoj od algoritma, programskog koda, izvršne datoteke do paketa namijenjenoga za distribuciju. Sve prikazati na Linux/Ubuntu platformi koristeći aplikacije otvorenog koda.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3Postignuti rezultati u odnosu na složenost zadatka: 2Jasnoća pismenog izražavanja: 3Razina samostalnosti: 3
Datum prijedloga ocjene mentora:	27.02.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 19.03.2017.

Ime i prezime studenta:	Tomislav Šantić
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	A4159, 22.09.2011.
Ephorus podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Programski jezik Python**

izrađen pod vodstvom mentora Doc.dr.sc. Damir Blažević

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija. Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

1.1 Zadatak završnog rada

Opisati povjesni razvoj, značajke i primjenu programskog jezika Python. Opisati sintaksu, dostupna razvojna okruženja i mogućnosti. Izraditi nekoliko aplikacija i prikazati njihov razvoj od algoritma, programskog koda, izvršne datoteke do paketa namijenjenoga za distribuciju također prikazati na Linux/Ubuntu platformi.

1.2 Povijesni razvoj programskog jezika Python

Python nastaje u kasnim osamdesetim godinama od strane nizozemskog programera Guido van Rossuma koji je imao viziju stvaranja novog jezika koji će biti jednostavan i ugodan za korištenje. Python za cilj ima kvalitetu, produktivnost, portabilnost i modularnost. Kvaliteta se postiže lakom čitljivošću i samom duljinom programskog koda, što implicira produktivnost jer programer mora pisati, ispravljati i održavati manje koda. Portabilnost je omogućena zbog toga što se Pyhton program može pokretati neovisno o operacijskom sutavu, a modularnost se postiže korištenjem raznih biblioteka koje se mogu iskoristiti za proširenje funkcionalnosti.

1.3 Instalacija Python unutar Linux okruženja

Obzirom da je Python programski jezik moguće podesiti na različitim operacijskim sustavima, instalacija je prikazana unutar Debian 8.6.0 distribucije Linuxa. Iako Python 2.7.x dolazi automatski podešen prilikom instalacije Debian operacijskog sustava, za Python 3.x.x. verziju potrebna je instalacija. U tablici 1.1. prikazani su koraci instalacije programskog jezika Python.

Tablica 1.1. *Koraci instalacije programskog jezika Python*

1. korak	Otvoriti terminal
2. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “su”
3. korak	Unesite zaporku “root” korisnika
4. korak	Upisati naredbu: “apt-get install build-essential libncursesw5-dev libreadline5-dev libssl-dev”
5. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “apt-get install libgdbm-dev libc6-dev libsqlite3-dev tk-dev”
6. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “wget http://www.python.org/ftp/python/python/3.x/Python-3.x.tar.bz2”

7. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “tar -xjf Python-3.xtar.bz2 cd Python-3.x”
8. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “./configure -prefix=/opt/python3”
9. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “make”
10. korak	Upisati sljedeću naredbu bez navodnika i pritisnuti tipku ENTER: “make install”

2. PYTHON

Obzirom da postoje brojni programski jezici, opravdano je pitanje: “Zašto Python?”. Python je jezik koji za cilj ima kvalitetu, produktivnost, portabilnost, modularnost i ugodnost korištenja. Kvaliteta se postiže lakom čitljivošću programskog koda, produktivnost se unaprijeđuje smanjenjem količine napisanog koda potrebnog za postizanje rezultata, portabilnost je omogućena zbog toga što se Python program može pokretati neovisno o operacijskom sustavu, a modularnost se postiže korištenjem raznih biblioteka koje se mogu iskoristiti za proširenje funkcionalnosti. Obzirom da je ugodnost korištenja subjektivna mjera, nužan je primjer kako bi se to moglo demonstrirati.

Sl. 2.1. *Generiranje Fibonaccijeve sekvence*

U primjeru programskog koda na slici 2.1. opisano je stvaranje funkcije naziva `fibonacci` koja generira Fibonaccijevu matematičku sekvencu i za parametar prima `unos` odnosno broj do kojeg se sekvenca generira, te rezultat pokretanja funkcije. U drugim programskim jezicima bilo bi nužno uvoditi dodatne bibloteke i module kako bi sličan kod mogao biti izvršen.

2.1. Varijable i tipovi varijabli

Varijable su memorijske lokacije koje služe za pohranu vrijednosti, što znači da se prilikom stvaranja nove varijable u memoriji rezervira određeni dio prostora za pohranu. Ovisno o tipu podatka, alocira se veći ili manji dio memorije i na taj način je moguće pohranjivati različite tipove podataka od kojih izdvajamo brojeve, znakove i polja znakova, no dakako postoje i kompleksniji tipovi podataka s kojima Python može raditi. Važno je naglasit da je sve u Pythonu objekt, odnosno tipovi podataka predstavljeni su kao objekti određenog tipa podatka.

2.1.1. Pridruživanje vrijednosti varijablama

Tipove varijabli u Pythonu nije potrebno eksplicitno pisati jer se deklaracija događa automatski prilikom upisivanja vrijednosti u varijablu. Znak jednakosti (=) koristi se za pridruživanje vrijednosti varijablama. S lijeve strane znaka jednakosti nalazi se varijabla dok s desne, njezina vrijednost. Primjer programskog koda na slici 2.2. prikazuje pridruživanje vrijednosti varijablama.

```
1  prirodni_broj = 100
2
3  udaljenost_u_miljama = 1372.37
4
5  ime_korisnika = "Perun"
6
7  # primjer višestrukog pridruživanja
8  ime_studenta, kolegiji, godina = "Pero", "Programiranje", "2"
9
```

Sl. 2.2. *Primjer pridruživanja vrijednosti*

2.1.2 Brojevi

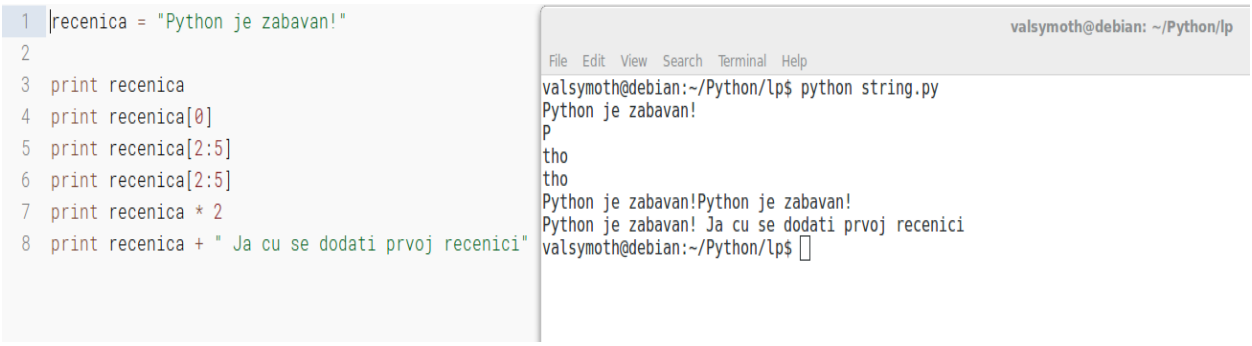
Brojevi su tipovi podataka koji se pohranjuju kao objekti numeričkog tipa. U tablici 2.1. prikazani su tipovi brojeva s s kojima je moguće raditi u programskom jeziku Python.

Tablica 2.1. *Tipovi brojeva*

INT	LONG	FLOAT	COMPLEX
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAE	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0j
-0x260	-052318172735L	-32.54e100	3e+26j
0x69	-4721885298529	70.2-E12	4.53e-7j

2.1.3. Stringovi

Kontinuirani nizovi znakova unutar navodnih znakova u Pythonu nazivaju se stringovima. Python omogućuje korištenje jednostrukih i dvostrukih navodnih znakova za označavanje strignova. Također su moguće operacije nad nizovima znakova pomoću operatora za zbrajanje i množenje.



Sl. 2.4. Operacije nad nizovima znakova

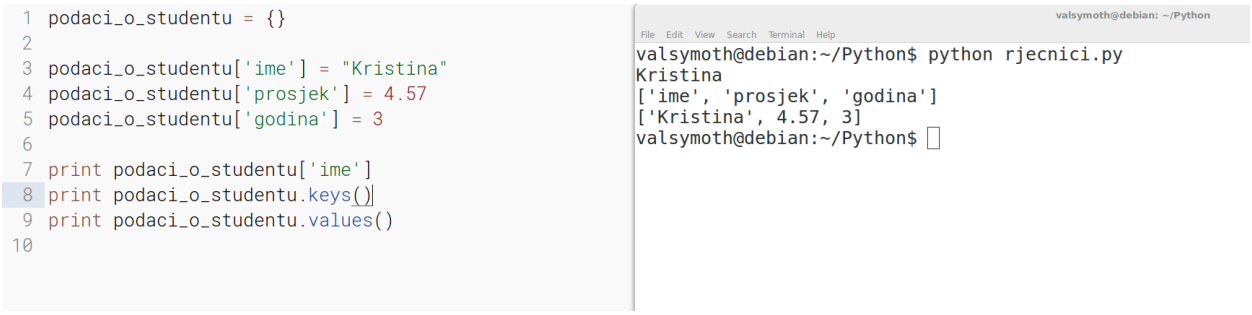
2.1.4. Liste

Liste predstavljaju složeni tip podatka. Liste sadrže podatke odvojene zarezom unutar uglatih zagrada ([ ]). Vrijednostima se može pristupati pomoću operatora odsjecanja, te je moguće i upravljanje pomoću operatora zbrajanja i množenja.

Sl. 2.5. *Operacije nad listama*

2.1.5. Rječnici

Rječnici u Python programskom jeziku funkcioniraju poput asocijativnih nizova u kojima postoje parovi ključa i vrijednosti. Rječnici se označavaju vitičastim zagradama ({}).



Sl. 2.6. *Operacije nad rječnicima*

2.2 Operatori

Operatori su jezični konstrukti koji omogućavaju modifikaciju vrijednosti operanda. Postoje aritmetički, relacijski, logički, i bit operatori kao i operatori pridruživanja.

2.2.1 Aritmetički operatori

U sljedećoj tablici prikazati ćemo aritmetičke operatore pretpostavljajući da varijabla A iznosi 10, a varijabla B iznosi 20.

Tablica 2.2. *Aritmetički operatori*

OPERATOR	OPIS	PRIMJER
+	Zbrajanje	A + B = 30
-	Oduzimanje	A – B = -10
*	Množenje	A * B = 200
/	Dijeljenje	B / A = 2
%	Modulo	A % B = 0
**	Eksponent	A**B = 10
//	Cjelobrojno djeljenje	9 // 2 = 4

2.2.2. Operatori uspoređivanja

U sljedećoj tablici prikazati ćemo operatore uspoređivanja pretpostavljajući da su A i B bilo koja dva realna broja.

Tablica 2.3. *Operatori uspoređivanja*

OPERATOR	OPIS	PRIMJER
==	Ako su vrijednosti jednake, uvjet postaje istinit	(a == b)
!=	Ako su vrijednosti nisu jednake, uvjet postaje istinit	A!=B
<>	Ako vrijednosti nisu jednake izraz postaje istinit. Sličan !=.	(a <> b)
>	Ako je vrijednost lijevog operanda veća od desnog, izraz je istinit.	A > B
<	Ako je vrijednost lijevog operanda manja od desnog, izraz je istinit.	A < B
>=	Ako je vrijednost lijevog operanda veća ili jednaka od desnog, izraz je istinit.	A >= B
<=	Ako je vrijednost lijevog operanda manja ili jednaka od desnog, izraz je istinit.	A <= B

2.2.3. Operatori pridruživanja

U sljedećoj tablici prikazati ćemo operatore pridruživanja s primjerima.

Tablica 2.4. *Operatori pridruživanja*

OPERATOR	OPIS	PRIMJER
=	Pridruživanje vrijednosti s desne strane operatora operandu s lijeva.	A = 3 + 7 Vrijednost A: 10
+=	Zbrajanje vrijednosti s obje strane operatora i pridruživanje lijevom operandu	A = 3 A += 7 Vrijednost A: 10
-=	Oduzimanje vrijednosti s lijeve strane operatora desnom i pridruživanje lijevom operandu	A = 7 A -= 3 Vrijednost A: 4
*=	Množenje vrijednosti s lijeve strane operatora desnom i pridruživanje lijevom operandu	A = 10 A *= 5 Vrijednost A: 50
/=	Dijeljenje vrijednosti s lijeve strane operatora desnom i pridruživanje lijevom operandu	A = 10 A /= 5 Vrijednost A: 2
%=	Modulus vrijednosti s lijeve strane operatora desnom i pridruživanje lijevom operandu	A = 10 A %= 5 Vrijednost A: 0
**=	Potenciranje vrijednosti s lijeve strane operatora desnom, te pridruživanje lijevom operandu	A = 10 A **= 2 Vrijednost A: 100
//=	Cjelobrojno dijeljenje vrijednosti s lijeve strane operatora desnom i pridruživanje lijevom operandu	A = 10 A //= 3 Vrijednost A: 3

2.2.4. Logički operatori

U sljedećoj tablici prikazati ćemo logičke operatore pretpostavljajući da su A i B bilo koja dva objekta unutar programskog jezika Python.

Tablica 2.5. *Logički operatori*

OPERATOR	OPIS	PRIMJER
AND	Logičko I	A and B
OR	Logičko ILI	A or B
NOT	Logičko NE	not A

2.2.5. Operatori identiteta

U sljedećoj tablici prikazati ćemo operatore istinitosti pretpostavljajući da su X i Y bilo koja dva objekta unutar programskog jezika Python.

Tablica 2.6. O*peratori identiteta*

OPERATOR	OPIS	PRIMJER
is	Vraća true ako su operatori jednaki objekti unutar memorije	X is Y
is not	Vraća false ako su operatori jednaki objekti unutar memorije	X is not Y

2.2.6. Prioriteti operacija

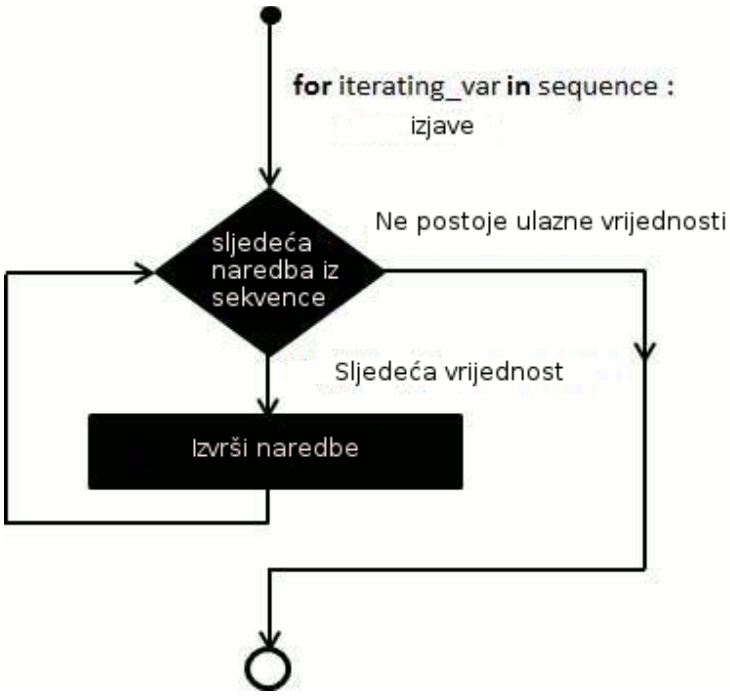
U sljedećoj tablici prikazani su prioriteti operacija počevši s najvažnijom, odnosno onom koja se prva izvodi, do posljednje.

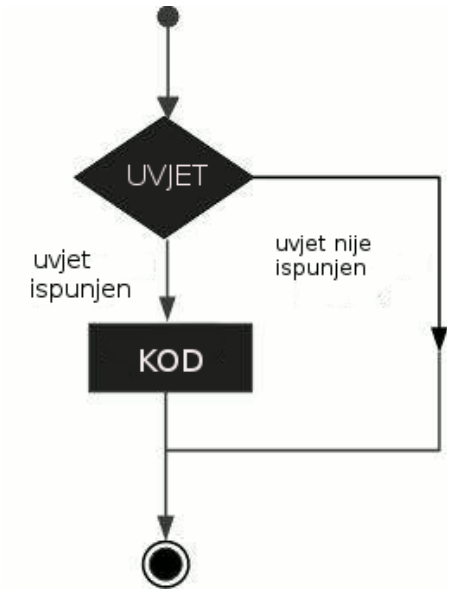
Tablica 2.7. *Prioriteti operacija*

OPERATOR	OPIS
**	Eksponent
~ + -	Komplement, unarni plus i unarni minus
* / % //	Množenje, djeljenje, modulo i cjelobrojno djeljenje
+ -	Plus, minus
>> <<	Pomicanje bitova u desno odnosno lijevo
&	Logičko I
^ |	EXILI ili ILI
<= < > >=	Operatori uspoređivanja

2.3. Odlučivanje

Odlučivanje unutar programskog jezika odnosi se na dio programskog koda u kojem se provjeravaju zadani izrazi te se na temelju toga obavlja grananje programa. Za bilo koji dani izraz moguće je reći kako može biti istinit ili lažan. Primjerce, izjavom da na kišoviti dan nosimo kišobran, za zadani dan ispituje se meteorološko vrijeme i temeljem rezultata tog ispitivanja donosi se odluka o nošenju ili ne nošenju kišobrana. Generalizirano odlučivanje prikazano je na primjeru slike 2.7.





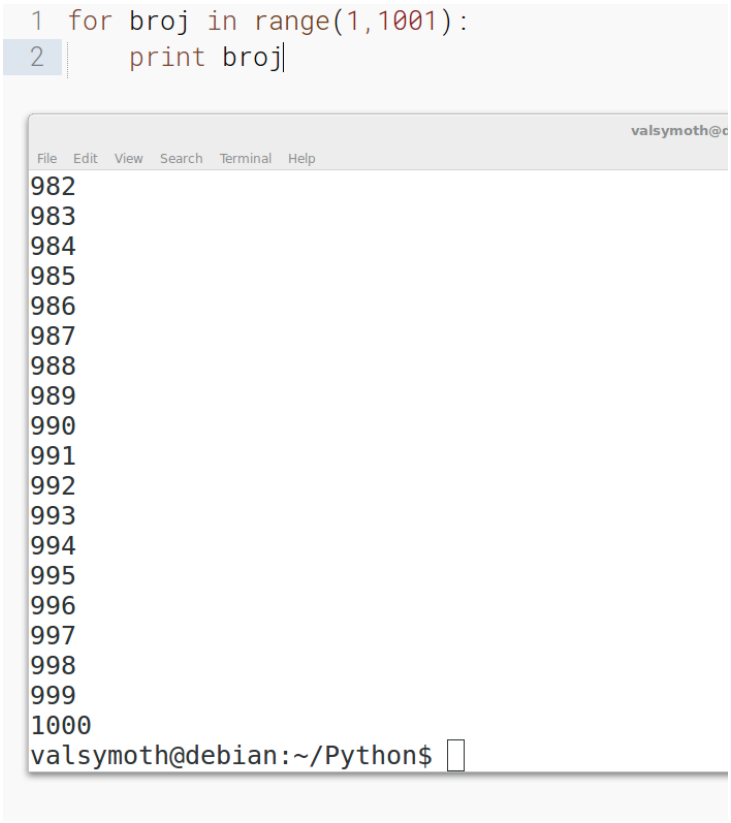
Sl. 2.7. *Odlučivanje*

U Pythonu odlučivanje moguće je pisati na sljedeći način:

```
kisovito_vrijeme = True
if ( kisovito_vrijeme == True):
      print “Danas pada kisa. Ponesite kisobran.”
```

2.4. Petlje

Petlje su apsolutno neophodne u programskim jezicima, i one omogućuju višestruko izvođenje programskog koda. Generalno, naredbe se izvode sekvencijalno i ponekad je potrebno jedan dio programskog koda izvršiti nekoliko puta. Primjerice, zadan je zadatak ispisa brojeva od 1 do 1000. Jedan, iako apsolutno redudantan način za obavljanje ove operacije bi bio da programer ručno unese 1000 brojeva. Vrijeme koje bi bilo potrebno za ručni unos 1000 brojeva je neprihvatljiv za bilo kakavo ozbiljno programiranje, stoga se problem rješava pomoću petlji na način prikazan na slici 2.8. Važno je napomenuti da postoji više načina za rješavanje ovog problema jer postoje različite vrste petlji .



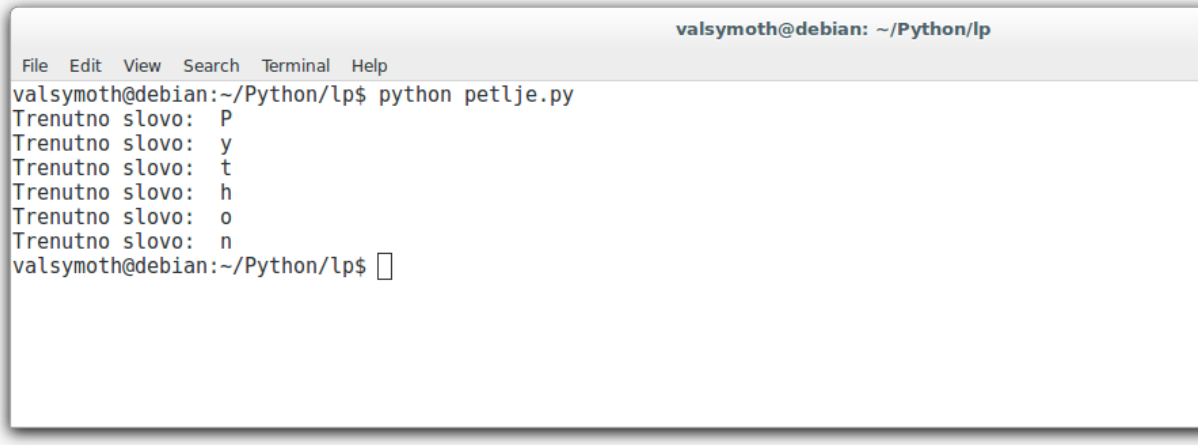
Sl. 2.8. *Ispis brojeva od 1 do 1000*

2.4.1. For petlja

Dijagram toka `for` petlje prikazan je na slici 2.9, a na slici 2.10. prikazan je programski kod i rezultat izvršavanja.

Sl. 2.9. *Dijagram toka for petlje*

```
1  for slovo in "Python":
2      print "Trenutno slovo: ", slovo
```



Sl. 2.10. *Primjer for petlje*

Na slici 2.11. prikazan je programski kod i rezultat izvršavanja indeksiranog ispisa for petlje.

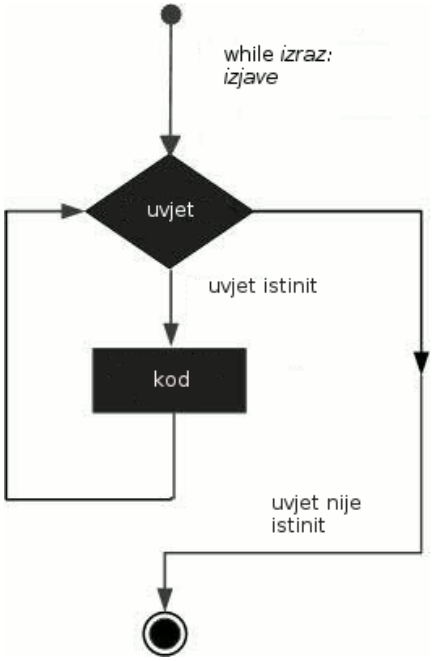
```
1  automobili = ["BMW", "Tesla", "Concept_One"]
2
3  for index in range(len(automobili)):
4      print "Trenutni automobil: ", automobili[index]
5
```

```
valsymoth@debian: ~/Python/lp
File  Edit  View  Search  Terminal  Help
valsymoth@debian:~/Python/lp$ python petlje.py
Trenutni automobil:  BMW
Trenutni automobil:  Tesla
Trenutni automobil:  Concept_One
valsymoth@debian:~/Python/lp$
```

Sl. 2.11. *Primjer indeksiranog ispisa for petlje*

2.4.2. While petlja

`While` petlja razlikuje se od `for` petlje jer će se ona izvršavati sve dok je uvjet istinit i dijagram toka prikazan je na slici 2.12.



Sl. 2.12. *Dijagram toka while petlje*

Na slici 2.13. vidljiv je programski kod i rezultat izvršavanja `while` petlje.

```
1  brojac = 0
2  while brojac < 5:
3      print "Brojac iznosi: ", brojac
4      brojac += 1 # jako važno!
```

```
valsymoth@debian: ~/Python/lp
File  Edit  View  Search  Terminal  Help
valsymoth@debian:~/Python/lp$ python petlje.py
Brojac iznosi:  0
Brojac iznosi:  1
Brojac iznosi:  2
Brojac iznosi:  3
Brojac iznosi:  4
valsymoth@debian:~/Python/lp$
```

Sl. 2.13. *Primjer while petlje*

Nužno je naglasiti kako je kod `while` petlje važno modificirati uvjet na takav način da se spriječi javljanje beskonačne petlje. Beskonačna petlja nastaje kada ispitivanje `while` uvjeta konstantno rezultira u logičkoj jedinici.

```
1  # beskonacna petlja
2
3  while True:
4      broj = raw_input("Unesite broj: ")
5      print broj
6  print "Ja se nikada necu ispisati"
```

```
valsymoth@debian: ~/Python/lp
File  Edit  View  Search  Terminal  Help
valsymoth@debian:~/Python/lp$ python petlje.py
Unesite broj: 1
1
Unesite broj: 2
2
Unesite broj: 3
3
Unesite broj: a
a
Unesite broj: -33
-33
Unesite broj: ^CTraceback (most recent call last):
  File "petlje.py", line 4, in <module>
    broj = raw_input("Unesite broj: ")
KeyboardInterrupt
valsymoth@debian:~/Python/lp$
```

Sl. 2.14. *Primjer beskonačne while petlje*

Posljednja linija programskog koda u navedenom primjeru nikada se neće izvoditi jer program ne izlazi iz `while` petlje. Kako bi izašli iz takvog programa nužno ga je prekinuti, specifično gledajući u Linuxima pritiskom tipke **CTRL + C.** Pritiskom tipke **CTRL + C** operacijski sustav šalje prekid i time zaustavlja rad programa.

2.5. Funkcije

Funkcije su blokovi organiziranog, ponovno iskoristivnog, koda koji ima zadatak obavljanje nekog složenijeg zadatka. Funkcije pružaju aplikacijama modularnost jer se funkcije mogu više puta iskoristiti unutar istog programa. Postoje ugrađene funkcije poput `raw_input()`, te korisničke funkcije.

2.5.1. Definiranje funkcija

Svaka funkcija u programskom jeziku Python mora započinjati ključnom riječi def nakon čega slijedi naziv funkcije te zatvorene zagrade tj. (). Funkcije mogu primati vrijednosti. stoga ukoliko funkcija prima neke vrijednosti one se navode unutar zagrada. Nakon zagrada obavezno slijedi znak dvotočja. Prva izjava nije nužna, te može poslužiti kao dokumentacija. Svaka sljedeća izjava mora biti ispravno uvučena te ići jedna ispod druge. Funkcija završava u trenutku kada razina identacije završi na nuli.

```
def ime_funkcije ( parametri ):
      “Dokumentacija funkcije”
      izjava 1
      izjava 2
      ...
      return [ izraz ]
```

2.5.2. Funkcijski poziv

Definicijom je funkciji dodijeljen naziv, parametri koje prima, kod koji se izvršava te povratni izraz. Kako bi se ta funkcija izvršila nužno ju je pozvati, a poziva se na način da se upiše naziv funkcije te da se navedu potrebni parametri. Na slici 2.15. možemo vidjeti primjer funkcijskog poziva.

Sl. 2.15. *Primjer funkcijskog poziva*

2.6. Klase i objekti

U objektno orijentiranim programskim jezicima klasom se nazivaju dijelovi programskog koda koji služe kao predlošci za stvaranje korisničkih tipova podataka. Objekt je instanca određene klase čije je ponašanje i inicijalno stanje definirano unutar klase. Na slici 2.16. prikazan je programski kod stvaranja klase i objekta, te rezultat izvršavanja.

```
1  class Zaposlenik:
2      "Klasa kojom predstavljamo zaposlenike"
3      broj_zaposlenika = 0
4
5      def __init__(self,ime,prezime,placa):
6          self.ime = ime
7          self.prezime = prezime
8          self.placa = placa
9          Zaposlenik.broj_zaposlenika += 1
10
11     def ispis_podataka(self):
12         print "Ime: ", self.ime, "\nPrezime: ", self.prezime, "\nPlaca:", self.placa, "\n"
13
14 zaposlenik1 = Zaposlenik("Ana","Leos",3000)
15 zaposlenik2 = Zaposlenik("Dunja","Toth",5000)
16
17 zaposlenik1.ispis_podataka()
18 zaposlenik2.ispis_podataka()
19
20
21
```

```
File  Edit  View  Search  Terminal  Help
valsymoth@debian:~/Python/lp$ python klase.py
Ime:  Ana
Prezime:  Leos
Placa: 3000

Ime:  Dunja
Prezime:  Toth
Placa: 5000

valsymoth@debian:~/Python/lp$
```

Sl. 2.16. *Primjer klase i objekta*

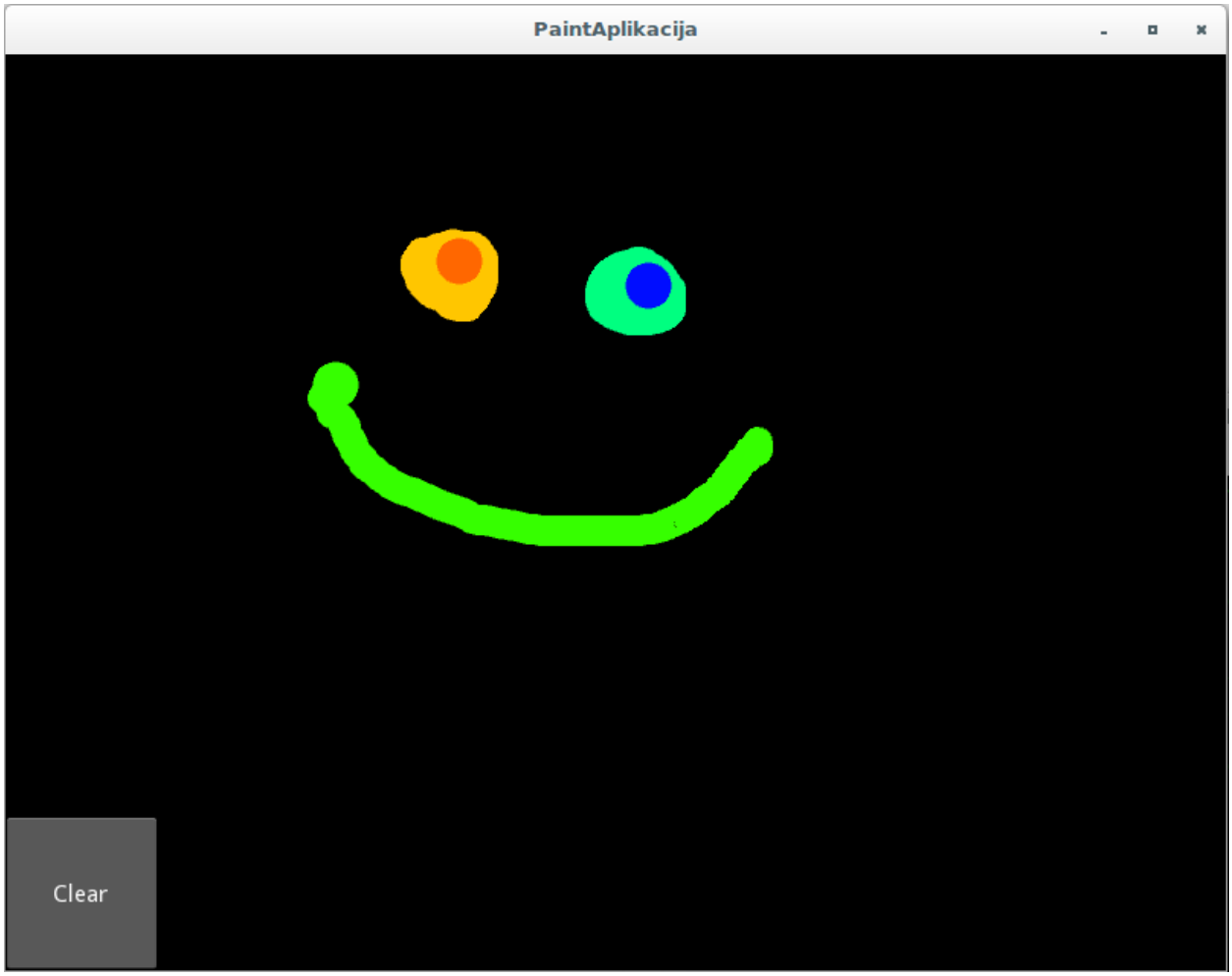
Varijabla `brojZaposlenika` je klasna varijabla čiju vrijednost dijele svi objekti klase `Zaposlenik`. Prva metoda `__init__()` je posebna metoda koja se naziva konstruktor ili inicijalizacijska metoda koja se poziva prilikom stvaranja novog objekta. Sve ostale metode unutar klase pišu se poput normalnih funkcija. Kako bi se stvorila instanca klase, tj. objekt, nužno je koristiti njegov naziv uz listu argumenata, kao što je u primjeru vidljivo: `Zaposlenik("Ana", "Leos", 3000)`.

3. PRIMJERI APLIKACIJA U PROGRAMSKOM JEZIKU PYTHON

U ovom poglavlju biti će prikazane aplikacije napisane u programskom jeziku Python koristeći Kivy framework. Kivy je besplatano (MIT licenca), više-platformsko (Linux, Windows, OS X, Android, iOS) rješenje za izradu grafičkih sučelja i odabran je zbog svoje jednostavnosti.

3.1. Kivy aplikacija za crtanje

Cilj ove jednostavne aplikacije jest omogućiti korisniku crtanje jednostavnijih crteža, uz mogućnost brisanje slike. Također bit će jasno vidljivo koliko malo linija koda je potrebno kako bi se napisala funkcionalna Kivy aplikacija. Programski kod biti će prikazan segmentno uz objašnjenje pojedinih dijelova, nakon čega će biti prikazan kod u cijelosti. Slika 3.1. prikazuje konačan izgled aplikacije za crtanje.



Slika 3.1 *Paint aplikacija*

Aplikacija započinje uvođenjem pomoćnih biblioteka i funkcija koje omogućavaju daljni rad:

```
from random import random
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.uix.button import Button
from kivy.graphics import Color, Ellipse, Line
```

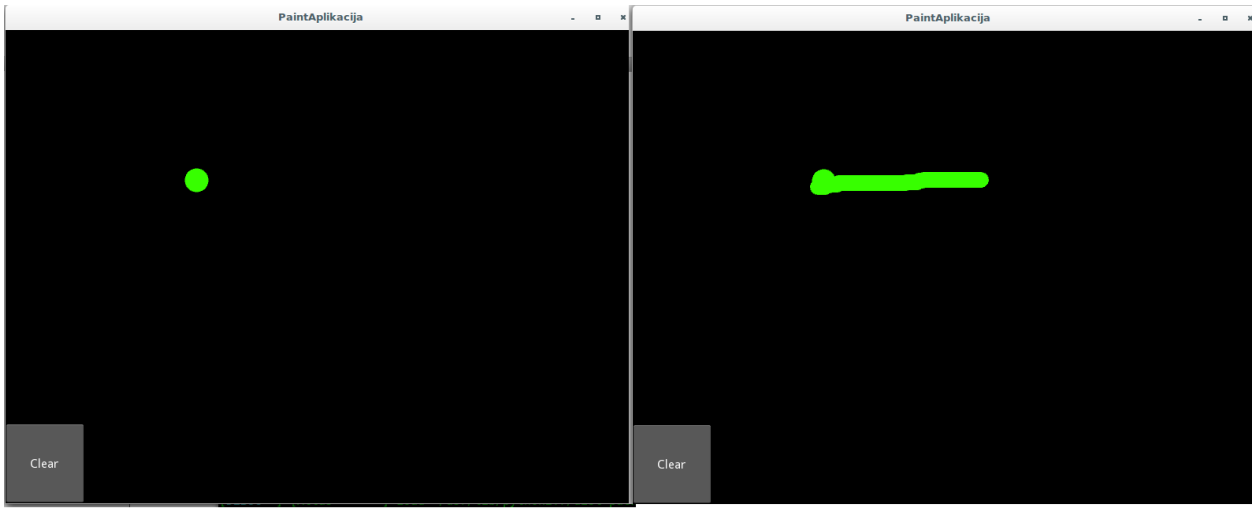
Svaka Kivy aplikacija može se gledati kao stablo **Widget** elemenata. Svaki **widget** element obuhvaća podatke, definira interakciju korisnika i ispisuje grafičke podatke. U našem slučaju definirali smo vlastiti **widget** naziva "MyPaintWidget" i pripadajuću korisničku interakciju:

```
class MyPaintWidget(Widget):

    def on_touch_down(self, touch):
        color = (random(), 1, 1)
        with self.canvas:
            Color(*color, mode='hsv')
            d = 30.
            Ellipse(pos=(touch.x - d / 4, touch.y - d / 4), size=(d, d))
            touch.ud['line'] = Line(points=(touch.x, touch.y), width = 10)

    def on_touch_move(self, touch):
        touch.ud['line'].points += [touch.x, touch.y]
```

Dvije metode navedene ovdje su `on_touch_down i on_touch_move.` Metoda `on_touch_down` definira što se događa u trenutku kada korisnik započne interakciju s aplikacijom što može biti mišem ili dodirom. U primjeru navedenom gore, definira se nasumična RGB vrijednost kao vrijednost boje i ispisuje se točka. U trenutku korisnikovog pomicanja dodirne točke, vrijednosti boje se upisuju na odgovarajuće koordinate i na taj način je dobivena linija. Na slici 3.2. prikazana je prvotna točka i linija u trenutko pomicanja.



Slika 3.2 *Paint aplikacija - točka i linija*

Sljedeću definirana klasa je `PaintAplikacijaApp`. Ovaj naziv je redudantan, no prilikom pokretanja aplikacije, Kivy koristi naziv prije App dijela kao naziv aplikacije. Unutar klase definiraju se `build` i `clear_canvas` metode. Unutar `build` metode definirana je jezgra aplikacije, poziv grafičkih elemenata te vraćanje podataka u grafičkom obliku. `clear_canvas` metoda omogućuje brisanje prethodnog crteža:

```
class PaintAplikacijaApp(App):

    def build(self):
        parent = Widget()
        self.painter = MyPaintWidget()
        clearbtn = Button(text='Clear')
        clearbtn.bind(on_release=self.clear_canvas)
        parent.add_widget(self.painter)
        parent.add_widget(clearbtn)
        return parent

    def clear_canvas(self, obj):
        self.painter.canvas.clear()
```

Naš program završava zaštitom od kolizije prilikom import funkcionalnosti i pokreće aplikaciju na sljedeći način:

```
if __name__ == '__main__':
    PaintAplikacijaApp().run()
```

3.2. Kivy kalkulator

Kao sljedeći primjer prikazujemo aplikaciju kalkulatora. U ovom primjeru prikazano je odvajanje prezentacijskog od aplikacijskog sloja što je baza svih naprednijih aplikacija koje se mogu napisati koristeći Kivy i Python. Programski kod biti će prikazan segmentno uz objašnjenje pojedinih dijelova. Na slici 3.3. prikazana je aplikacija u izvođenju.

Slika 3.3 *Kalkulator*

Aplikacija započinje uvođenjem pomoćnih biblioteka i funkcija koje nam omogućuju daljni rad:

```
import kivy
kivy.require('1.8.0')
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
```

GridLayout omogućava korištenje responzivne grafičke matrice s kojom je definirano samo sučelje aplikacije. Klasa `CalcGridLayout( GridLayout )` predstavlja grafičko sučelje i u njoj su definirane metode koje se koriste unutar aplikacije.

```
class CalcGridLayout(GridLayout):

      def calculate(self,calculation):
            if calculation:
                  try:
                        self.display.text = str(eval(calculation))
                  except Exception:
                        self.display.text = "Error"
```

Metoda `calculate` izračunava konačni izraz koji je spremljen unutar tekstualnog polja. Izraz `str(eval(calculation))` koristi se radi evaluacije numeričkog izraza i pretvorbu u podatak tipa `string`. Rezultat je prikazan unutar glavnog polja za ispis rezultata. Sljedeći korak je definiranje same aplikacije i njezino pokretanje:

```
class CalculatorApp(App):
      def build(self):
            return CalcGridLayout()

calcApp = CalculatorApp()
calcApp.run()
```

Cijelu aplikaciju pohranjena je kao datoteku naziva `main.py` Prvi dio aplikacije uspješno je napisan. Sljedeći korak je napisati datoteku koja će definirati grafičko sučelje same aplikacije. U Kivyu grafičkoj datoteci dodjelimo ekstenziju .kv. Na taj način se povezuje Python aplikacija s grafičkim sučeljem definiranim u .kv datoteci. U primjeru prve aplikacije to nije učinjeno obzirom da je aplikacija jednostavna, no bilo koja kompleksnija aplikacija mora slijediti ovaj način odvajanja aplikacijskog od prezentacijskog sloja jer bi u protivnom bila narušena čitljivost aplikacijskog koda.

```
# Definiranje korisnicke tipke
<CustButton@Button>:
    font_size: 32

#Dodjela naziva(id) za lakse referenciranje
<CalcGridLayout>:
    id: calculator
    display: entry
    rows: 5
    spacing: 10
    padding: 10


    # Mjesto na kojem se rezultati ispisuju
    BoxLayout:
        TextInput:
            id: entry
            multiline: False
            font_size: 32


    # kada je tipka pritisnuta, promijeni vrijednost
    BoxLayout:
        spacing: 10
        CustButton:
            text: "7"
            on_press: entry.text += self.text
        CustButton:
            text: "8"
            on_press: entry.text += self.text
        CustButton:
```

Slika 3.4 *calculator.kv prvi dio koda*

```
        CustButton:
            text: "9"
            on_press: entry.text += self.text
        CustButton:
            text: "+"
            on_press: entry.text += self.text

    BoxLayout:
        spacing: 10
        CustButton:
            text: "4"
            on_press: entry.text += self.text
        CustButton:
            text: "5"
            on_press: entry.text += self.text
        CustButton:
            text: "6"
            on_press: entry.text += self.text
        CustButton:
            text: "-"
            on_press: entry.text += self.text

    BoxLayout:
        spacing: 10
        CustButton:
            text: "1"
            on_press: entry.text += self.text
        CustButton:
            text: "2"
            on_press: entry.text += self.text
```

Slika 3.5 *calculator.kv drugi dio koda*

```
61          CustButton:
62              text: "3"
63              on_press: entry.text += self.text
64          CustButton:
65              text: "*"
66              on_press: entry.text += self.text
67
68      # poziv metode/funkcije u trenutku pritiska znaka "="
69      BoxLayout:
70          spacing: 10
71          CustButton:
72              text: "AC"
73              on_press: entry.text = ""
74          CustButton:
75              text: "0"
76              on_press: entry.text += self.text
77          CustButton:
78              text: "="
79              on_press: calculator.calculate(entry.text)
80          CustButton:
81              text: "/"
82              on_press: entry.text += self.text
```

Slika 3.6 *calculator.kv treći dio koda*

3.3 Distribucija i stvaranje .deb datoteke

Deb je format programskih datoteka na Debian Linux distribuciji. Debian paketi su standardne Unix ar datoteke koje sadrže dvije tar arhive. Jedna sadrži kontrolne informacije, dok druga sadrži podatke koje se instaliraju. **Dpkg** (Debian Package Manager) je alat pomoću kojeg se obavlja instalacija .deb datoteka iako krajnji korisnici najčešće nemaju interakciju s dpkg već s APT menadžerom paketa poput Synaptic-a.

Postupak stvaranja .deb datoteke:

Tablica 3.1 *Koraci stvaranja .deb datoteke*

1. korak	Otvaranje terminala i stvaranje datoteke pomoću naredbe: “mkdir kalkulator_1.0-1”
2. korak	Stvaranje kontrolnih direktorija pomoću naredbi: “mkdir kalkulator_1.0-1/usr” “mkdir kalkulator_1.0-1/usr/local” “mkdir kalkulator_1.0-1/usr/local/bin”
3. korak	Kopiranje programa u direktoriji “ kalkulator_1.01/usr/local/bin”
4. korak	Stvaranje “Debian” datoteke pomoću naredbe: “mkdir kalkulator_1.0-1/DEBIAN”
5. korak	Stvaranje “control” datoteke upisujući naredbu: “vim kalkulator_1.0-1/DEBIAN/control “ i upisujemo unutar datoteke: Package: Kalkulator Version: 1.01 Section: base Priority: optional Architecture: i386 Depends: nekaBiblioteka (>= 1.2.13), sljedećaBiblioteka(>= 1.2.6) Maintainer: Vaše Ime <you@email.com> Description: Kalkulator
6. korak	Pakiranje u “.deb” datoteku pomoću naredbe: “dpkg-deb --build helloworld_1.01”

4. ZAKLJUČAK

Moderni programski jezici današnjice moraju efikasno stvarati aplikacije sutrašnjice. Zahtjevi za kvalitetom, optimalnim radom, skalabilnošću i sigurnošću modernih programa sve su veći. Suvremeni programer danas mora na takve zahtjeve odgovarati odabirom skupa alata koji mu neće stajati na putu ostvarivanja tih zadataka. Python predstavlja jednu od mogućih opcija koje programer može uključiti u svoj razvojni proces. Kvaliteta Python koda je konceptualno integrirana u sam dizajn jezika i time se ostvaruje optimalni rad, jer se s manjim opsegom programskog koda može postići puno više nego u klasičnim jezicima "niže razine". Sigurnost danas je važnija nego ikada prije, no razvojnim procesom jezika koji se temelji na "open source" principima to se uistinu lako postiže. Od jednostavnih početničkih aplikacija do znanstveno-istraživačkih kalkulacija, Python svojom skalabilnošću pogoduje svima. Guido van Rossum imao je veliku viziju stvaranja programskog jezika koji će predstavljati budućnost, i ta vizija je danas zasigurno ostvarena. Python je programski jezik koji će se nastaviti razvijati i utjecati na buduće programere na nezamislive načine, a budućnost mu se čini sve sjajnijom obzirom na nadolazeće potrebe "Internet of Things" i "Big Data" aplikacija.

5. LITERATURA

[1] Z. A. Shaw, Learn Python The Hard Way, Addison - Wesley Professional, 11. listopad 2013.

[2] M. Lutz, Learning Python, O'Reilly Media, 6. srpanj 2013.

[3] A. Sweigart, Automate the Boring Stuff with Python: Practical Programming for Total Beginners, No Starch Press, 1. svibanj 2015

[4] L. Ramalho, Fluent Python: Clear, Concise, and Effective Programming, O'Reilly Media, 20. kolovoz 2015.

[5] E. Matthes Python Crash Course: A Hands-On, Project-Based Introduction to Programming, No Starch Press, 30. studeni 2015.

[6] Sužbena stranica programskog jezika Python [Mrežno]. Dostupno: https://www.python.org/ [Pokušaj pristupa 22 9 2016]

[7] Kivy službena stranica i dokumentacija [Mrežno]. Dostupno: https://kivy.org/docs/api-kivy.html [Pokušaj pristupa 22 9 2016]

[8] Tutorialpoint pregled Python programskog jezika. [Mrežno]. Dostupno: http://www.tutorialspoint.com/python/ [Pokušaj pristupa 22 9 2016]

[9] Stackoverflow pitanja i odgovori u vezi programskog jezika Python. [Mrežno]. Dostupno: http://stackoverflow.com/questions/tagged/python [Pokušaj pristupa 22 9 2016]

[10] Codeschool pregled Python programskog jezika. [Mrežno]. Dostupno: https://www.codeschool.com/learn/python [Pokušaj pristupa 22 9 2016]

[11] CodeAcademy pregled Python programskog jezika. [Mrežno.] Dostupno: https://www.codecademy.com/learn/python [Pokušaj pristupa 22 9 2016]

[12] Sololern pregled Python programskog jezika. [Mrežno.] Dostupno: https://www.sololearn.com/Course/Python/ [Pokušaj pristupa 22 9 2016]

[13] LearnXinYminutes pregled Python programskog jezika. [Mrežno.] Dostupno: https://learnxinyminutes.com/docs/python/ [Pokušaj pristupa 22 9 2016]

[14] CodingBat pregled Python programskog jezika. [Mrežno.] Dostupno: http://codingbat.com/python [Pokušaj pristupa 22 9 2016]

[15] PlanetPython pregled Python programskog jezika. [Mrežno.] Dostupno: http://planetpython.org/ [Pokušaj pristupa 22 9 2016]

SAŽETAK

Naslov: PROGRAMSKI JEZIK PYTHON

Ideja programskog jezika Python nastaje u kasnim osamdesetim godinama od strane nizozemskog programera Guido van Rossuma koji je imao viziju stvaranja jezika koji će imati mogućnosti rukovanja iznimkama i služiti kao moguće sučelje Amoeba operacijskom sustavu. U veljači 1991. godine van Rossum objavljuje prvu verziju programskog jezika (verzija 0.9.0) s podrškom za objektno orijentirano programiranje i ugrađenim tipovima podataka poput list, dict, str i drugim. 16. 8. 2000. godine objavljuje se verzija 2.0. s brojnim novim dodacima od kojih su značajni ciklično detektirajući upravljač memorijom te podrška za Unicode znakovni standard.

Ključne riječi: Python, Guido van Rossum, programiranje Python, Kivy, sintaksa

ABSTRACT

Title: PROGRAMMING LANGUAGE PYTHON

The idea for Python programming language was conceived in late 1980's and its implementation begun in December 1989 by a Dutch programmer Guido van Rossum in Netherlands. Python is a programming language developed with a purpose of improving programmers productivity, by means of portability, modularity and ease of use. Quality is achieved with Python's easy to read syntax, productivity is improved by reducing the lines of code necessary to write to achieve a specific goal. Portability and modularity are result of Python's open source implementation and a vast number of libraries that extend Python language.

Keywords: Python, programming language, Guido van Rossum, Python, Kivy syntax

ŽIVOTOPIS

Tomislav Šanitć, rođen 21. 4. 1993. u Vinkovcima. Pohađao osnovnu školu "Antun Gustav Matoš" nakon čega upisuje ekonomsku i trgovačku školu Ivana Domca, također u Vinkovcima. Po završetku srednješkolskog obrazovanja upisuje Elektrotehnički fakultet u Osijeku. Uz studiji bavi se izradom i dizajnom web stranica.

PRILOG A. Izvorni kod aplikacije za crtanje

```
from random import random
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.uix.button import Button
from kivy.graphics import Color, Ellipse, Lineclass MyPaintWidget(Widget):

    def on_touch_down(self, touch):
        color = (random(), 1, 1)
        with self.canvas:
            Color(*color, mode='hsv')
            d = 30.
            Ellipse(pos=(touch.x - d / 4, touch.y - d / 4), size=(d, d))
            touch.ud['line'] = Line(points=(touch.x, touch.y), width = 10)

    def on_touch_move(self, touch):
        touch.ud['line'].points += [touch.x, touch.y]

class PaintAplikacijaApp(App):

    def build(self):
        parent = Widget()
        self.painter = MyPaintWidget()
        clearbtn = Button(text='Clear')
        clearbtn.bind(on_release=self.clear_canvas)
        parent.add_widget(self.painter)
        parent.add_widget(clearbtn)
        return parent

    def clear_canvas(self, obj):
        self.painter.canvas.clear()if __name__ == '__main__':
    PaintAplikacijaApp().run()
```