SVEUČILIŠTE/UNIVERZITET „VITEZ“ TRAVNIK

FAKULTET **INFORMACIJSKIH TEHNOLOGIJA**

STUDIJ I CIKLUS; GODINA STUDIJA**: I CIKLUS**; **I GODINA**

SMIJER: **INFORMACIJSKE TEHNOLOGIJE**



**DUVNJAK BENJAMIN**

**PROGRAMSKI JEZIK PYTHON**

**SEMINARSKI RAD**

Travnik, 23.01.2020 godine

SVEUČILIŠTE/UNIVERZITET „VITEZ“ TRAVNIK

FAKULTET **INFORMACIJSKIH TEHNOLOGIJA**

STUDIJ I CIKLUS; GODINA STUDIJA**: I CIKLUS**; **I GODINA**

SMIJER: **INFORMACIJSKE TEHNOLOGIJE**

**PROGRAMSKI JEZIK PYTHON**

**SEMINARSKI RAD**

IZJAVA:

Ja **Benjamin Duvnjak**, student Sveučilišta/Univerziteta „Vitez“ u Travniku, Index broj: **161-19/DIIT** odgovorno i uz moralnu i akademsku odgovornost izjavljujem da sam ovaj rad izradio potpuno samostalno uz korištenje citirane literature i pomoć profesora.

Potpis studenta:

STUDENT: **Benjamin Duvnjak**

PREDMET: **Principi programiranja**

PROFESOR:**Dr.sc. Mahir Zajmović**

ASISTENT:  **NEMA**

SADRŽAJ

[1. UVOD 1](#_Toc30776142)

[1.1 PROBLEM PREDMET I OBJEKT ISTRAŽIVANJA 1](#_Toc30776143)

[1.2 SVRHA I CILJEVI ISTRAŽIVANJA 1](#_Toc30776144)

[1.3 RADNA HIPOTEZA I POMOĆNE HIPOTEZE 1](#_Toc30776145)

[1.4 NAUČNE METODE 1](#_Toc30776146)

[1.5 STRUKTURA RADA 1](#_Toc30776147)

[2. PROGRAMSKI JEZIK PYTHON 2](#_Toc30776148)

[2.1 OSNOVNA SVOJSTVA JEZIKA PYTHON 3](#_Toc30776149)

[2.2 USPOREDBA S DRUGIM PROGRMASKIM JEZICIMA 5](#_Toc30776150)

[3. POKRETANJE PYTHON PROGRAMA 5](#_Toc30776151)

[3.1 ISPIS/UNOS 7](#_Toc30776152)

[4. TIPOVI PODATAKA 8](#_Toc30776153)

[4.1 BROJEVI 9](#_Toc30776154)

[4.2 STRINGOVI 9](#_Toc30776155)

[4.3 OPERATORI 10](#_Toc30776156)

[5. FUNKCIJE 10](#_Toc30776157)

[5.1 LOKALNE I GLOBALNE VARIJABLE 11](#_Toc30776158)

[5.2 PODRAZUMIJEVANI ARGUMENTI 12](#_Toc30776159)

[6. PYTHON MODULI 13](#_Toc30776160)

[6.1 SYS 13](#_Toc30776161)

[6.2 OS 14](#_Toc30776162)

[6.3 SOCKET 15](#_Toc30776163)

[6.4 MATH I RANDOM 17](#_Toc30776164)

[6.5 KORISNIČKI MODULI 18](#_Toc30776165)

[6.6 C à PYTHON 19](#_Toc30776166)

[7. STRUKTURE PODATAKA 21](#_Toc30776167)

[7.1 LISTA 21](#_Toc30776168)

[7.2 TUPLE 22](#_Toc30776169)

[7.3 RIJEČNIK 22](#_Toc30776170)

[8. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE 23](#_Toc30776171)

[8.1 PYTHON PODRŽAVA NASLJEĐIVANJE 25](#_Toc30776172)

[9. RAD S TEKSTUALNIM DATOTEKAMA 26](#_Toc30776173)

[9.1 PICKLE 27](#_Toc30776174)

[10. RUKOVANJE IZNIMKAMA 28](#_Toc30776175)

[11. USPOREDBA DVA PROGRAMSKA JEZIKA(PHP I PYTHON) 30](#_Toc30776176)

[12. ZAKLJUČAK 31](#_Toc30776177)

[13. LITERATURA 32](#_Toc30776178)

[Knjige: 32](#_Toc30776179)

[Linkovi: 32](#_Toc30776180)

# UVOD

## PROBLEM PREDMET I OBJEKT ISTRAŽIVANJA

U ovom radu ću objasniti šta je to „programski jezik PYTHON“ kao i njegovu upotrebu.

## SVRHA I CILJEVI ISTRAŽIVANJA

Svrha istraživanja u ovom radu, je da se prikaže šta je zapravo „PYTHON“ i kako on funkcioniše.

## RADNA HIPOTEZA I POMOĆNE HIPOTEZE

***Na temelju problema i predmeta istraživanja postavlja se glavna (radna) hipoteza:***

***„****Prevedeni jezik višeg nivoa,veoma jednostavan i syntksom lahkom za naučiti.Podržava module i pakete koji ga čine modularnim i code-reused,a zbog velikog broja istih je jedan od najpopularniji programskih jezika danas.****“***

## NAUČNE METODE

***U ovom seminarskom radu ću koristiti sljedeće znanstvene (stručne) metode:***

* Metoda analize i sinteze,
* Metoda klasifikacije,
* Metoda modeliranja.

## STRUKTURA RADA

Struktura seminarskog rada je usklađena sa uputstvom za pisanje seminarskog rada na prvom ciklusu studija kao i temi seminarskog rada. On sadrži pet poglavlja. Prvo poglavlje, uvod, sadrži pet pod poglavlja:

1. Problem, predmet i objekt istraživanja,
2. Svrha i ciljevi istraživanja,
3. Radna hipoteza i pomoćne hipoteze,
4. Znanstvene metode,
5. Struktura rada.

# PROGRAMSKI JEZIK PYTHON

Oko 1972. godine pojavio se jezik C, prvi jezik opće namjene. Sedamnaest godina kasnije Guido van Rossum, istraživač na CWI (Stichting Mathematisch Centrum) u Nizozemskoj, radi na projektima vezanima uz interpreterskim jezik ABC. Nezadovoljan nefleksibilnošću tog i sličnih programskih jezika i nedostatkom alata za automatiziranje dosadnih sistemskih problema, odlučuje se razviti novi programski jezik. Želio je programski jezik koji bi mu omogućio rješavanje generalnih administratorskih zadaća brzo i fleksibilno, ali je istovremeno želio iskoristiti snagu sistemskih poziva Amoeba distribuiranog OS-a. I tako je, krajem 1989. godine, počeo razvijati programski jezik Python, koji je nazvao prema kultnoj BBC seriji „[[1]](#footnote-1)Monty Python's leteći cirkus“. 1996. godine Van Rossum je napisao o nastanku Python-a:  
"Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)."  
(Introduction to Programming Python, by Mark Lutz, published by O'Reilly)   
Iako je u početku zamislio jezik koji bi bio specifičan samo za Amoeba OS, brzo je shvatio da mu je potreban jezik koji bi se mogao pokretati na svakoj platformi. 1991. godine u open source zajednicu je pušten interpreter za novi programski jezik, Python, napisan u C-u. Zbog činjenice da je Python open source aplikacija, danas se uspješno može vrtiti na gotovo svakoj platformi (Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, Palm OS, Acorn RISC OS, PlayStation, Sharp Zaurus, Pocket PC, …)



Slika 1. Logo Python programskog jezika

## OSNOVNA SVOJSTVA JEZIKA PYTHON

Python je interpreterski jezik, što znači da se program ne kompajlira tj. ne prevodi se u strojni kod procesora nego u poseban, tzv. Python binarni kod. (Python bytecode) kojeg  pokreće Python virtualna mašina. Zato se programi vrlo jednostavno mogu prenositi sa stroja na stroj, dovoljno je da se kod kopira i pokrene. Naravno, pod uvjetom da na tom stroju postoji odgovarajući virtualni stroj. Zbog činjenice da se Python kod interpretira, a ne izvodi izravno u strojnom jeziku optimiziranom za odgovarajući procesor, Python programi su sporiji od ekvivalentnih programa pisanih u drugim jezicima. Ali, kao i Java, i Python virtualni strojevi pružaju mogućnost prevođena Python binarnog koda u matični izvedbeni kod koji je puno bliži strojnom kodu i kao takav omogućava i brže izvođenje. Tako Python omogućava dobre perfomanse svojim programima, a u isto vrijeme zadržava snagu interpreterskih jezika.

Python podržava proceduralno i objektno orijentirano programiranje. Kod proceduralnog programiranja program se rastavlja na niz manjih zadataka – procedura – koje su dovoljno jednostavne da bi se mogle izraziti pomoću naredbi programskog jezika. U takvom načinu programiranja procedure se odvajaju od podataka koje obrađuju. Za razliku od toga, u objektno orijentiranom programiranju, cilj je spajanje podataka i operacija nad njima u objekte. Programer koji će koristiti taj objekt jednostavno od objekta zatraži određenu uslugu i ne zamara se razmišljajući o načinu na koji objekt funkcionira.  
Modularnost. Python je moguće koristiti na više načina. Najčešće se kod rastavlja na više manjih modula koji međusobno mogu "komunicirati". Može se vrlo jednostavno iz jednog modula pozivati procedure ili koji koristiti objekte drugog modula.. Što je još važnije, pristup modulima je uvijek isti, bez obzira da li se radi o već ugrađenim modulima iz Python-ove standardne biblioteke ili o modulu koji ste napravili trenutak prije. Python-ova standardna biblioteka podržava regularne izraze, generiranje dokumentacije, threadove, baze podataka, web pretraživače, CGI, ftp, elektroničku poštu, XML, HTML, WAV datoteke, kriptografiju, GUI (grafičko sučelje) upotrebom [[2]](#footnote-2)Tk i još mnogo toga. Isto tako, moduli se mogu pisati i u drugim jezicima, pa se mogu napisati u C-u, kompajlirati i preko Python sučelja, koje je potpuno isto kao i kod Python modula, jednostavno koristiti. To omogućuje da se dijelovi koda koji predstavljaju usko grlo (vremenski su zahtjevni) ubrzaju i time znatno poboljšaju cjelokupno izvršavanje programa. Takav način programiranja se podrazumijeva u jezicima kao što su C i C++, ali je iznimka među skriptnim jezicima.

U python-u je vrlo jednostavno otkriti pogrešku u izvršavanju programa. U slučaju pogreške u izvođenju programa, moguće je pristupiti stogu programa, "stack trace" će točno odrediti vrstu i mjesto greške (exception) što omogućuje vrlo lagano lociranje i ispravljanje pogreške u programu.  
Automatsko skupljanje smeća. Curenje memorije ili prerano uništenje objekata u tradicionalnim jezicima je vrlo čest uzrok "blokiranja" rada programa ili cijelog računala. Python interpreter sam se brine da se oslobodi memorija koja više nije potrebna – mehanizam za sakupljanje smeća (garbage collection) je ugrađen u sustav i automatski se pokreće po potrebi. Programeri tako ne moraju imati pod kontrolom gdje su rezervirali prostor za neki objekt i gdje će se taj prostor osloboditi, već se mogu fokusirati na samo rješavanje problema. S druge strane, kod automatskog skupljanja smeća, sustav sam procjenjuje kada treba to sakupljanje započeti i nerijetko se to događa upravo onda ka program obavlja neku zahtjevnu operaciju. Zato se i kod ugrađenog automatskog sustava za skupljanje smeća treba često paziti da ne novi objekti ne stvaraju neracionalno.

## USPOREDBA S DRUGIM PROGRMASKIM JEZICIMA

Python se vrlo često uspoređuje s drugim programskim jezicima zbog činjenice da podržava više programskih paradigmi, sam Python je nastao kao kombinacija mnogih drugih programskih jezika, uključujući ABC, Modula-3, C, C++, Algol-68, SmallTalk, razne skriptne jezike. Ali, pošto je Python interpreterski jezik, najviše je usporedbi sa Perl-om, Javom, Tcl-om i JavaScript-om.  Perl, kao i Python, nudi snagu potpunog programskog jezika kao i pristup sistemskim pozivima. Najveća snaga Perl-a je u radu sa stringovima, Perl je najmoćniji upravo u radu sa regularnim izrazima, filtriranju teksta i još uvijek se najčešće koristi za pisanje CGI skripti. Problem Perl-a je u kompliciranoj sintaksi koju je ponekad vrlo teško čitati, što dovodi do frustrirajuće teškog učenja jezika i nerazumljivih programa.

Python se vrlo često, zbog  podržavanja objektno orijentiranog programiranja i slične sintakse, uspoređuje sa Javom. Java, iako puno jednostavnija od C++-a, još uvijek je kruta, posebno ako se želi napisati mali, brzi program. Python-ova jednostavnost omogućuje puno brže pisanje malih programa nego upotreba čiste Jave. Veliki korak u odnosu Python-a i Jave je razvoj JPython-a, Python interpretera napisanog isključivo u Javi. Sada je moguće pokretati Python programe na strojevima na kojima se vrti samo Java virtualni stroj. Vrlo je jednostavno u JPython-u raditi s Java objektima, komunicirati s njima, koristiti Java biblioteke klasa, kao da je Java u Python bila integrirana od početka razvoja jezika.

Od jezika koji je najčešće uspoređuju s Python-om najsličniji je JavaScript po svojoj sintaksi i po činjenici da je objektno orijentiran. Programeri JavaScript-e vrlo brzo i bez problema prihvaćaju Python. Python se, za razliku od JavaScripte, pokreće izvan Web pretraživača, a omogućava sistemske pozive i podržava vrlo jednostavno rješavanje problema koji se tradicionalno obavljaju pomoću shell skripta.

# POKRETANJE PYTHON PROGRAMA

Tri su načina pokretanja Python programa.Najjednostavniji je način startanjem interpretera interaktivno. Pokreće se iz komandne linije a kod se unosi liniju po liniju i izvršava. Dodatne opcije kada se interpreter pokreće iz komande linije.

-d  
provide debug output  
-O  
generate optimized bytecode (resulting in .pyo files)  
-S  
do not run import site to look for Python paths on startup  
-v  
verbose output (detailed trace on import statements)  
-X  
disable class-based built-in exceptions (just use strings);   
obsolete starting with version 1.6  
-c cmd  
run Python script sent in as cmd string  
file  
run Python script from given file (see below)

Primjer 2.1. (interaktivni interpreter iz komandne linije):

$ python

Python 2.4 (#4, Dec 10 2004, 19:19:57)

[GCC 2.95.2 20000220 (Debian GNU/Linux)] on linux2

Type "copyright", "credits" or "license" for more information.

IDLE 1.1 -- press F1 for help

>>> print 'hello world'

hello world

>>>

Drugi način je pokretanje skripte napisane u Python-u. Interpreter se poziva zajedno sa skriptom. Većina skripta završava sa ekstenzijom .[[3]](#footnote-3)py. Vrlo je važno da se dobro izabere editor u kojem će se pisati program. Više editora koji podržavaju python sintaksu, među njima su najpopularniji i najmoćniji Vim i Emacs.  
  
Primjer 2.2. (hello world skripta):

#!/usr/bin/pyhon

# Filename : hello\_world.py

print 'hello world'

</pre>

/usr/bin/pyhon označava puni put prema python interpreteru.

Pokretanje interpretera sa skriptom:

$ python hello\_world.py

hello world

U Unix-u je moguće automatski pokrenuti interpreter bez da ga pokrenemo iz komadne linije. Jednostavno postavimo modove tako da skripta postane executable.

$ chmod a+x hello\_world.py

$ ./hello\_world.py

hello world

Treći način pokretanja Python programa je korištenje grafičkom sučelja iz IDE (integrated development environment) okoline.  
IDLE je prvi Unix IDE za Python. Razvio ga je Guido van Rossum, prvi put za Python 1.5.2. IDLE je baziran na Tk-u, što zahtijeva da ima Tcl/Tk instaliran na stroju. Većine IDE sučelja sadrži i editor i interpreter.

## ISPIS/UNOS

Unos korisnika je vrlo jednostavno dobiti pomoću raw\_input() funkcije.  
Primjer 2.3. Unošenje teksta:

>>> ime = raw\_input('Unesi ime: ')

Unesi ime: ana

>>> print 'Tvoje ime je:', ime

Tvoje ime je: ana

Primjer 2.3. je samo za unos teksta. Kada želimo raditi s cijelim brojevima, unošeni tekst se mora konvertirati u integer pomoću funkcije int()

Primjer 2.4. Ispis broja:

>>> broj = raw\_input('unesi broj: ')

unesi broj: 45

>>> print 'kvadrat unesenog broja je: %d' %int(broj)\*\*2

kvadrat unesenog broja je: 2025

Isto kao i C, Python u naredbi print ima format specifikatore, koji počinju sa %, ima %d za integere, %s za stringove i %f za realne brojeve.  
Komentar započinje direktno iza znaka # i nastavlja se do kraja linije.

# TIPOVI PODATAKA

Postoje dva načina provjere tipa varijabli u programi. Za vrijeme kompajliranja programa (static typing) ili za vrijeme rada programa (dynamic typing). Python prodržava dynamic type što znači da nema deklaracija tipa varijabli već da sam program za vrijeme rada odlučuje koji će tip pridružiti kojoj varijabli. Takav način korištenja varijabli je Dave Thomas (razvojna zajednica Ruby jezika) nazvao duck typing ("(referring to a value) if it walks like a duck, and quacks like a duck, then it is a duck.").

U praksi, dinamičko deklariranje funkcionira tako da svaki program za vrijeme svog rada pridjeljuje varijablama tipove ovisno o tome koje su im vrijednosti pridružene (operator pridruživana u Python-u je =). Valjanost tih tipova se provjerava tek kada se te varijable koriste u operacijama – u tom trenutku se i generiraju greške u slučaju upotrebe operatora koji nisu dozvoljeni nad tipovima varijabli koje su u izrazu.

Među jezicima koji podržava dinamičko deklariranje, Python je umjereno [[4]](#footnote-4)type-checked. Impilicitno pretvaranje je definirano za numeričke tipove, npr. dozvoljeno je množiti complex broj i long integer bez eksplicitnog pretvaranja. Ali nema implicitnog pretvaranja između numeričkih tipova i stringova (za razliku od Perl-a, u Python se ne može operaciji sa stringom pridružiti broj.)  
Ono što Python ima je ugrađena funkcija eval() koja čita string kao moguću operaciju:

>>> eval('5+6')

11

>>> eval('7\*8')

56

Vrlo je važno je zapamtiti da je Python osjetljiv na velika i mala slova pa varijabla brojac nije jednaka varijabli Brojac.

## BROJEVI

Python podržava četiri tipa brojeva: integer, long integer (mogu biti prezentirani i oktalno i heksadecimalno), float (realni) i complex (kompleksni).  
  
Primjeri unosa:

int 0101 84 -237 0x80 -0X92

long 29979062458L -84140l 0xDECADEDEADBEEFBADFEEDDEAL

float 3.14159 4.2E-10 -90. 6.022e23 -1.609E-19

complex  6.23+1.5j -1.23-875J 0+1j  9.80665-8.31441J -.0224+0j

## STRINGOVI

String je u Python-u neprekidni niz karaktera između navodnika (jednostrukih ili duplih).Evo nekoliko primjera korištenja operacija sa stringovima.

>>> pystr = 'Python'

>>> iscool = 'is cool!'

>>> pystr[0]

'P'

>>> pystr[2:5]

'tho'

>>> iscool[:2]

'is'

>>> iscool[3:]

'cool!'

>>> iscool[-1]

'!'

>>> pystr + iscool

'Pythonis cool!'

>>> pystr + ' ' + iscool

'Python is cool!'

>>> pystr \* 2

'PythonPython'

U Python-u se uvijek može doći do brze pomoći za korištenje neke funkcije ili naredbe. Jednostavno se u interpreter prompt-u napiše help(naredba).  
Npr. za pomoć u radu sa stringom:  
>>>  help(str)

## OPERATORI

Standardni operatori funkcioniraju u Python-u kao i u bilo kojem drugom programskom jeziku.   
+          -          \*         /         %         \*\*  
 Uz standardno zbrajanje, oduzimanje, množenje, cjelobrojno dijeljenje/ostatak cjelobrojnog dijeljenja Python nudi i operator za potenciranje \*\*.  
Python podržava operatore poput \*=, +=, -=, /=, ali ne podržava C-ovsko predinkrementiranje i postinkrementiranje.  
Isto tako, Python ima i standardne operator za uspoređivanje.  
<          <=       >         >=        ==       !=  <>  
Za različito postoje dva operatora != i <>. Preporuča se korištenje operatora !=.  
Isto tako podržani su logički operatori and, or i not. Upotrebljavajući logičke operatore možemo stvarati izraze poput:

>>> (2 < 4) and (2 == 4)

False

>>> (2 > 4) or (2 < 4)

True

>>> not (6.2 <= 6)

True

>>> 3 < 4 < 5

True

Zadnji izraz je u Python-u legalan, i zapravo je kraći zapis ovoga.

>>> (3 < 4) and (4 < 5)

True

# FUNKCIJE

Ponavljani postupak, blok naredbi koji se često koristi, stavlja se u jednu zasebnu cjelinu, funkciju i svaki put kad zatreba pokreće pozivom te funkcije. Onom tko poziva funkciju nije važno što u njoj piše, jedino je važno ono što funkcija prima (argumenti) i ono što funkcija vraća.  
  
Primjer 5.1. Prva funkcija:

#!usr/bin/python

#prva\_funkcija.py

#definiranje funkcije nadiMax sa dva argumenta a i b

def nadiMax(a,b):

    if a > b:

        return a

    else:

        return b

#poziv funkcije

print nadiMax(4,7), 'je najveci'

x=-3

y=-6

print nadiMax(x,y),'je\_najveci'

$ python prva\_funkcija.py

7 je najveci

-3 je najveci

Odmah nakon return naredbe vraća se tamo od kud je funkcija pozvana. Svaka funkcija implicitno sadrži return naredbu. Ako uz return naredbu ništa nije navedeno funkcija će vratiti None. None je specijalna vrijednost koja predstavlja ne postojanje vrijednosti. Kada varijabla nema vrijednosti, ima vrijednost None.

## LOKALNE I GLOBALNE VARIJABLE

Pojedina funkcija u svojem radu može za realizaciju željenog algoritma trebati niz pomoćnih varijabli (objekata)

#!usr/bin/python

#funkcija\_lokalna.py

def funkcija(x):

    print 'Lokalna varijabla x je',x

    x = 2

    print 'Lokalna varijabla x je',x

x=50

funkcija(x)

print 'x je',x

$ python funkcija\_lokalna.py

Lokalna varijabla x je 50

Lokalna varijabla x je 2

x je 50

Python podržava i globalne objekte. Ako se želi u funkciji koristi varijabla definirana izvan nje koristi se global.

#!usr/bin/python

#funkcija\_globalna.py

def funkcija():

    global x

    print 'x prije promjene je',x

    x = 2

    print 'x poslije promjene je',x

x=50

funkcija()

print 'x je',x

$ python funkcija\_globalna.py

x prije promjene je 50

x poslije promjene je 2

x je 2

## PODRAZUMIJEVANI ARGUMENTI

U slučaju da se neke funkcije pozivaju vrlo često sa istim vrijednostima mogu se definirati podrazumijevane vrijednosti argumenata funkcije. Argumenti s podrazumjevanom vrijednošću moraju se nalaziti na kraju liste argumenata. Isto tako umjesto pozicije u pozivu funkcije vrijednosti se mogu vezati uz neku određenu varijablu.

#!usr/bin/python

#default\_argumenti.py

def funkcija(a,b=5,c=10):

    print 'a je',a,',b je',b,', c je',c

#dozvoljeni pozivi

funkcija(3,7)

funkcija(25,c=24)

funkcija(c=50,a=100)

$ python default\_argumenti.py

a je 3 ,b je 7 , c je 10

a je 25 ,b je 5 , c je 24

a je 100 ,b je 5 , c je 50

# PYTHON MODULI

Osnovna namjera modula je omogućiti efikasnije programiranje. Moduli su dokumenti koji sadrže definicije i implementacije funkcija. Tako se onda mogu grupirati funkcije u module i koristiti iz svakog python programa. Te funkcije se mogu pozivati iz bilo koje programa tako da se moduli "importaju". Modul je zapravo python program koji sadrži funkcije i varijable koje je pisac modula definirao. Python nudi svoju standardnu biblioteku, koja dolazi sa svakom instalacijom interpretera, u obliku modula. Sadrži vrlo veliku količinu korisnih modula za rad sa gotovo svime što se može zamisliti.   
Većina funkcija za rad sa sistemom dolazi u dva modula: sys (koji podržava rad sa Python interpreterom) i os (koji podržava rad sa operacijskim sustavom na kojem se Python interpreter vrti).

## SYS

Primjer 6.1. Upotreba sys.argv liste (sadrži argumente komandne linije)

#!/usr/bin/python

# argv.py

import sys

# Funkcija koja ispisuje fajl na standardni izlaz

def readfile(filename):

    f = file(filename)

    while True:

        line = f.readline()

        if len(line) == 0:

            break

        print line,

    f.close()

#glavni program

if len(sys.argv) < 2:

    print 'nema argumenta'

    sys.exit()

if sys.argv[1].startswith('--'):

    option = sys.argv[1][2:]

    #dohvaca prvi argument i kopira string osim prva dva znaka

    if option == 'verzija':

        print 'Verzija 1.00'

    elif option == 'help':

        print 'Ovo je help. Ovaj program ispisuje dokument na standardni izlaz'

        print 'opcije programa ukljucuju'

        print '--verzija - ispisuje verziju'

        print '--help - ispisuje pomoc i izlazi'

else:

    for filename in sys.argv[1:]:

        readfile(filename)

Model se uključuje koristeći import naredbu. U ovom primjeru importali smo sys modul. U trenutku kad Python interpreter dođe do linije u kojoj piše import sys traži dokument sys.py u nekom od direktorija koji su u path-u. Ako nađe dokument, naredbe u glavnom bloku tog modula se izvršavaju i funkcije modula se nakon toga mogu pozivati iz programa.

## OS

Primjer 6.2. Izlistavanje sadržaja direktorija

>>> for line in os.popen('ls /B').readlines(  ):

    print line[:-1]

Primjer 6.3. (backup dokumenata i direktorija)

#!/usr/bin/python

#  backup\_version.py

import os, time

# dokumenti i direktoriji za koje se radi backup

izvor = ['/home/g2swaroop/all', '/home/g2swaroop/bin']

# odredisni direktorij

odr\_dir = '/mnt/d/backup/'

# datum – stvara poddirektorij u glavnom odredisnom direktorju

danas = odr\_dir + time.strftime('%Y%m%d')

# The time - the name of the zip archive

sad = time.strftime('%H%M%S')

# komenat korisnika

komentar = raw\_input('Enter a comment --> ')

if len(komentar) == 0:     # da li je komentar unesen

 # ime zip dokumenta

odr = danas + os.sep + sad + '.zip'

else:

odr = danas + os.sep + sad + '\_' + \

komentar.replace(' ', '\_') + '.zip'

# ako nema poddirektorija, kreiraj ga

if not os.path.exists(danas):

os.mkdir(danas)

print 'Uspjesno je kreiran direktorij ', danas

# zip koji se pokrece

zip\_naredba = 'zip -qr %s %s' % (odr, ' '.join(izvor))

# pokreni backup

if os.system(zip\_naredba) == 0:

print 'Uspjesni backup u', odr

else:

print 'Stvaranje backupa je propalo.'

## SOCKET

Da se stvori veza između mašina, Python programi importaju socket modul, kreiraju socket objekt, i pozivaju metode tog objekta da otvore vezu i šalju i primaju podatke. Metode socketa se direktno mapiraju na socket pozive u C biblioteci. Sljedeci primjer čeka vezu na socketu, i kroz socket vraća ono što je primio, dodajući [[5]](#footnote-5)'Echo=>' ispred stringa.  
  
Primjer 6.4. Socket na server

from socket import \*                  # ukljuci modul socket

myHost = ''                           # server, '' znaci localni host

myPort = 50007                        # slusaj na slobodnom portu

sockobj = socket(AF\_INET, SOCK\_STREAM)       # napravi TCP socket objekt

sockobj.bind((myHost, myPort))               # postavi ga na port

sockobj.listen(5)                            # slusaj, dozvoli 5 konekcija

                           # cekanju

while 1:                                     # slusaj dok traje proces

    connection, address = sockobj.accept()   # cekaj na sljedecu konekciju

    print 'Server connected by', address     # konekcija je novi socket

    while 1:

        data = connection.recv(1024)    # citaj sljedecu liniju na klijentu

        if not data: break

        connection.send('Echo=>' + data)     # posalji odgovor klijentu

     connection.close()

Na klijentskoj strani, upotrebljavaju se socketi pomoću kojih se šalju podaci na server, i ispisuje odgovor servera; 'localhost' znači da se klijent i server vrte na istoj mašini što nam omogućava da testiramo skriptu. Za testiranje preko interneta: pokreni server na drugoj mašini, i postavi serverHost ili argv[1] na IP adresu te mašine.  
  
Primjer 6.5. socket na klijentu

import sys

from socket import \*

serverHost = 'localhost'          # server ime

serverPort = 50007                # slobodni port koji koristi server

message = ['Hello network world']        # tekst koji se salje na server

if len(sys.argv) > 1:

    serverHost = sys.argv[1]       # ili server iz prvog argumenta

    if len(sys.argv) > 2:                # ili iz drugog argumenta komandne linije

        message = sys.argv[2:]           # poruka za svaki argument

sockobj = socket(AF\_INET, SOCK\_STREAM)   # napravi TCP/IP socket objekt

# spoji se na server i port servera

sockobj.connect((serverHost, serverPort))

for line in message:

    sockobj.send(line)                   # posalji preko socketa liniju

    data = sockobj.recv(1024)       # primi liniju od servera preko socketa

    print 'Client received:', `data`

sockobj.close()                        # zatvori socket, server prima eof

Pošto je importanje modula vremenski zahtjevno Python optimizira modul i stvara Python binarni kod sa ekstenzijom pyc. Taj optimizirani Python-ov binarni kod ne ovisi o platformi na kojoj se vrti i može se koristiti i iz drugih programa.  
Svaki modul ima ime koje se može dobiti sa naredbom \_\_name\_\_ , koje je postavljen na \_\_main\_\_ kada se modul poziva samostalno, što se može iskoristit ako se želi izbjegnuti pokretanje nekog dijela glavnog programa u slučaju da je pozvan kao modul.

if \_\_name\_\_=='\_\_main\_\_':

    print 'samo ako se sam pokrecem'

    print 'a ne kao modul iz drugog programa'

## MATH I RANDOM

>>> import math

>>> math.cos(math.pi / 4.0)

0.70710678118654757

>>> math.log(1024, 2)

10.0

>>> import random

>>> random.choice(['apple', 'pear', 'banana'])

'apple'

>>> random.sample(xrange(100), 10)   # sampling without replacement

[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]

>>> random.random()    # random float

0.17970987693706186

>>> random.randrange(6)    # random integer chosen from range(6)

4

## KORISNIČKI MODULI

Svaki Python program je ujedno i modul. Tako da su svi primjeri u seminaru do sada moduli.  
Primjer 6.6. Moj prvi modul

#name.py

if \_\_name\_\_=='\_\_main\_\_':

    print 'samo ako se sam pokrecem'

verzija='1.01'

def funkcija(a,b):

    return (a\*b)

#moj\_modul\_primjer.py

import name

print name.verzija

print name.funkcija(4,7)

$ python moj\_modul\_primjer.py

1.01

28

ili se može eksplicitno navesti koje se funkcije importaju.  
  
Primjer 6.7. Moj drugi modul

#moj\_modul\_primjer1.py

from name import funkcija

print funkcija(4,7)

$ python moj\_modul\_primjer1.py

28

Funkcija dir() lista sve funkcije i varijable uključene u modul.

>>> import name

>>> dir(name)

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', 'funkcija', 'verzija']

## C à PYTHON

Kada u programu postoje dijelovi koda koji predugo traju i usporavaju rad programa  python omogućava korištenje modula napisani i kompajliranih u C-u, što uvelike može ubrzati kritične dijelove koda.  
  
Primjer  6.8. Hello.c  
  
/\* Jednostavan C modul za Python, "hello" \*/

#include <Python.h>

#include <string.h>

/\* module funkcija \*/

static PyObject \*                          /\* vraća objekt \*/

message(PyObject \*self, PyObject \*args)    /\* self neiskorišten u modulu \*/

{

    /\* args poziva iz Pythona \*/

    char \*fromPython, result[64];

    /\* pretvori Python -> C \*/

    if (! PyArg\_Parse(args, "(s)", &fromPython))

        /\* null=exception \*/

    return NULL;

    else {

        strcpy(result, "Hello, ");

    /\* dodaje string poslan iz Python-a \*/

        strcat(result, fromPython);

    /\* pretvaranje Python -> c \*/

    return Py\_BuildValue("s", result);

          }

}

/\* registracijska tablica  \*/

static struct PyMethodDef hello\_methods[] = {

/\* ime metode, pointer na funkciju, uvijek-tuple \*/

    {"message", message, 1},        {NULL, NULL}

};

,

/\* inicijalizacija modula \*/

/\* poziva se na prvi import \*/

void inithello(  )

{

/\* ime je važno jedino ako se dinamično učitava u Python \*/

    (void) Py\_InitModule("hello", hello\_methods);

}

#makefile.hello

PY = $(DIR\_PYTHON\_HEADER)

hello.so: hello.c

    gcc hello.c -g -I$(PY)/Include -I$(PY) -fpic -shared -o hello.so

clean:

    rm -f hello.so core

$ make –f makefile.hello

$ Python

>>> import hello                                   # import C modula

>>> hello.message('svijet')                         # poziv C funkcije

'Hello, svijet'

>>> hello.message('Ana')

'Hello, Ana'

C funkcije izgledaju i koriste se iz Python-a kao da su napisane u njemu. Vrlo je malo razlika između C modula hello i Python modula osim po imenu (ne završava na .py ili .pyc). C moduli čak i odgovaraju na [[6]](#footnote-6)dir() funkciju.

>>> dir(hello)                                     # C modul atributi

['\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', 'message']

>>> hello.\_\_name\_\_, hello.\_\_file\_\_

('hello', './hello.so')

>>> hello.message                                  # C funkcijski objekt

# STRUKTURE PODATAKA

Strukture podataka služe spremanju povezanih podataka u kolekciju. Tri su ugrađene strukture podataka u Python-u – lista, tuple i rječnik.

## LISTA

Lista je struktura podataka koja sadrži uređenu kolekciju, tj. u listu možeš dodavati niz objekata koji su međusobno povezani. U listu možeš dodavati, iz nje brisati ili ju pretraživati.

#!usr/bin/python

# lista.py

#Ovo je moja lista imena

imena=['haris','ajdin','muhamed','nihad','emina']

print 'U list je',len(imena),'imena'

print 'imena:',

for ime in imena:

    print ime,

print '\ndodavanje amine'

imena.append('amina')

print 'moja imena su sada',imena

imena.sort()

print 'sortirana imena su,',imena

print 'prvo ime je',imena[0]

brisano=imena[0]

del imena[0]

print 'obrisano je',brisano

print 'moja imena su sada',imena

$ python lista.py

U list je 5 imena

imena: haris ajdin muhamed nihad emina

dodavanje amine

moja imena su sada ['haris', 'ajdin', 'muhamed', 'nihad', 'emina', 'amina']

sortirana imena su, ['amina', 'nihad', 'muhamed', 'ajdin', 'emina', 'haris']

prvo ime je amina

obrisano je amina

moja imena su sada ['nihad', 'ajdin', 'haris', 'emina', 'haris']

Još nekoliko metoda liste:   
append(x) – dodaje na kraj liste  
insert(i,x) – dodaje element x na poziciju i  
remove(x) – briše prvi element iz liste  
pop(i)  - briše element na poziciji i, i vraća ga  
index(x) – vraća indeks prvog elementa u listi s vrijednošću x  
count(x) – brojac pojavljivanja elementa x u listi  
sort() – sortira elemente u listi  
reverse() – okrece redosljed elemenata u listi

## TUPLE

Tuple ima istu strukturu kao i lista,  s tom razlikom da se elementi u tuple ne mogu mijenjati. Koriste se kada je sigurno da se lista objekata u programu neće mijenjati.

#!/usr/bin/python

# tuple.py

zoo = ('vuk','slon','lisica')

print 'Broj zivotinja je',len(zoo)

novi\_zoo = ('zebra','zeba',zoo)

print novi\_zoo

print novi\_zoo[2]

print novi\_zoo[2][2]

$ python tuple.py

Broj zivotinja je 3

('zebra', 'zeba', ('vuk', 'slon', 'lisica'))

('vuk', 'slon', 'lisica')

lisica

## RIJEČNIK

Rječnik je kao telefonski imenik gdje se može naći broj osoba znajući njihovo ime. Svakom objektu u rječniku pridružuje se ključ sa vrijednostima. Ključ mora biti jedinstven – ne može se naći prava osoba ako ih ima više sa istim ključem. Za ključeve se može koristiti samo nepromjenjive objekte poput stringova, dok se za vrijednosti može koristiti sve.

#!/usr/bin/python

# dict.py

tel={'mario':'3245-543',

     'Ajdin':'4324-323',

     'Haris':'4324-432'

     }

print 'telefon Ajdin-a je %s' % tel['Ajdin']

tel['Mirza']='4323-112'

del tel['Ajdin']

print '%d je ljudi u imeniku.' % len(tel)

for ime,telefon in tel.items():

    print 'ime %s - tel %s' %(ime,telefon)

if tel.has\_key('Mirza'):

    print 'Mirzin telefon je %s' %tel['Mirza']

$ python dict.py

telefon Ajdin-a je 4324-323

3 je ljudi u imeniku.

ime mario - tel 3245-543

ime Haris - tel 4324-432

ime Mirza - tel 4323-112

Mirzin telefon je 4323-112

# OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Za razliku od proceduralnog programiranja koje je orijentirano na funkcije, objektno orijentirano programiranje je orijentirano na objekte, strukture koje kombiniraju podatke sa funkcionalnošću.   
U Python programskom jeziku sve je objekt. To uključuje i funkcije i sve ugrađene tipove podataka pa čak i integer.

Metode klada se razlikuju od običnih funkcija jedino po tome da imaju posebnu varijablu imena self – ta varijabla odnosi se upravo na objekt koji sadrži te metode. Self varijabla je ekvivalenta this pokazivaču u C++-u.  
U klasi se može definirati specijalna metoda imena \_\_init\_\_ koja će se pokrenuti odmah nakon što se objekt stvori. Koristi se za inicijaliziranje početnih vrijednosti ako je potrebno. Odgovara konstruktoru u C++-u.  
  
Postoji još specijalnih metoda:  
  
\_\_del\_\_ (self)  
poziva se u prije nego što se objekt uništi (destruktor)  
\_\_str\_\_(self)  
poziva se kada koristimo print sa objektom ili kada se upotrebljava str()  
\_\_lt\_\_(self,other)  
Poziva se kada se koristi operator <. Postoje metode za sve operatore (+,>,…)  
\_\_getitem\_\_(self,key)  
Poziva se kada se koristi operator []  
x[key]  
\_\_len\_\_(self)  
Poziva se kada se built-in funkcija len() koristi za objekt

Unutar objekata postoje dva tipa varijabli – varijable klase i varijable objekta. Razlika između tih dviju vrsta je u vlasništvu – pitanje je tko je vlasnik tih varijabli – objekt ili klasa?  
Varijable klase se dijele između svih objekata (instanci) klase. Varijable objekta su specifične za pojedinačni objekt i ne dijele se između objekata.

#!/usr/bin/python

# objvar.py

class Osoba:

    populacija=0

    def \_\_init\_\_(self,ime):

        'Inicijalizacija osobe.'

        self.ime = ime

        print('inicijalizacija %s') %self.ime

        'kada je osoba kreirana populacija se poveca'

        Osoba.populacija += 1

    def ispisImena(self):

        print 'moje ime je %s' %self.ime

    def kolikoIma(self):

        #uvijek je barem jedna

        if Osoba.populacija ==1:

            print 'jedina sam osoba ovdje'

        else:

            print 'ima %s ljudi.' %Osoba.populacija

zagor = Osoba('Zagor')

zagor.ispisImena()

zagor.kolikoIma()

haris = Osoba('Haris')

haris.ispisImena()

haris.kolikoIma()

marija = Osoba('Marija')

marija.ispisImena()

marija.kolikoIma()

$ python objvar.py

inicijalizacija Zagor

moje ime je Zagor

jedina sam osoba ovdje

inicijalizacija Haris

moje ime je Haris

ima 2 ljudi.

inicijalizacija Marija

moje ime je Marija

ima 3 ljudi.

## PYTHON PODRŽAVA NASLJEĐIVANJE

#!/usr/bin/python

# nasljedivanje.py

class ClanSkole:

    'Bilo koji clan skole'

    def \_\_init\_\_(self, ime,godine):

        self.ime = ime

        self.godine = godine

        print 'inicijalizirani clan je %s' %self.ime

    def ispis(self):

        print 'Ime: %s, Godine:%s' %(self.ime,self.godine)

class Ucitelj(ClanSkole):

    'Svi ucitelji'

    def \_\_init\_\_(self,ime,godine,placa):

        ClanSkole.\_\_init\_\_(self,ime,godine)

        self.placa = placa

        print 'inicijaliziran ucitelj %s' %self.ime

    def ispis(self):

        ClanSkole.ispis(self)

        print 'Placa je %s' %self.placa

class Ucenik(ClanSkole):

    'svi studenti'

    def \_\_init\_\_(self,ime,godine,ocjene):

        ClanSkole.\_\_init\_\_(self,ime,godine)

        self.ocjene = ocjene

        print 'inicijaliziran ucenik %s' %self.ime

    def ispis(self):

        ClanSkole.ispis(self)

        print 'Ocjene su %s' %self.ocjene

u = Ucitelj('Gosp. Grahorac',56,232321)

uc = Ucenik('Benjamin',23,84)

print

clanovi=[u,uc]

for clan in clanovi:

    clan.ispis()

    #radi i za ucenike i za ucitelje

$ python nasljedivanje.py

inicijalizirani clan je Gosp. Grahorac

inicijaliziran ucitelj Gosp. Grahorac

inicijalizirani clan je Benjamin

inicijaliziran ucenik Benjamin

Ime: Gosp. Grahorac, Godine:56

Placa je 232321

Ime: Benjamin, Godine:23

Ocjene su 84

# RAD S TEKSTUALNIM DATOTEKAMA

Mogućnost kreiranja, čitanja i ispisivanja tekstualnih datoteka je vrlo važan dio mnogih programa.Datoteke se mogu otvarati za čitanje ili pisanje tako da se prvo kreira objekt tipa file. Tada se koriste read, readline, ili write metode file objekta da se čita ili piše u datoteku ovisno o tome u kojem je modu otvorena datoteka. Nakon završetka rada s datotekom metodom close se ona zatvara.  
  
Primjer 9.1 Tekstualna datoteka

#!/usr/bin/python

# fileobj.py

tekst='''\

Ovo je prva linija tekst.

ovo je druga linija teksta.

Ovo je treca linija teksta.

'''

f = file('tekst.txt','w')

f.write(tekst)

f.close

f=file('tekst.txt') #datoteka se otvara u read modu

while True:

    linija=f.readline()

    if len(linija) == 0: #duzina 0 indicira kraj fajla

        break

    print linija

f.close()

$ python fileobj.py

Ovo je prva linija tekst.

ovo je druga linija teksta.

Ovo je treca linija teksta.

## PICKLE

Standardni modul koji omogućava da se bilo koji Python objekt sprema u dokument i dobavlja kasnije (trajno spremanje objekata). Postoji još jedan modul cPikle koji se ponaša kao i pickle modul osim što je napisan u C-u pa time puno brži. Korištenje i jednog i drugog modula je potpuno identično.  
  
Primjer 9.2. Pickling i unpickling

#!/usr/bin/python

# pickle.py

import cPickle

listaDok = 'podaci.data' #ime dokumenta koji koristimo za spremanje

lista=['hrast','bukva','lipa']

#pisanje, spremanje podataka

f = file(listaDok,'w')

cPickle.dump(lista,f)

f.close()

del lista

#citanje podataka

f = file(listaDok)

spremljena = cPickle.load(f)

print spremljena

$ python pickle.py

['hrast', 'bukva', 'lipa']

# RUKOVANJE IZNIMKAMA

>>> 10 \* (1/0)

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

ZeroDivisionError: integer division or modulo by zero

>>> 4 + spam\*3

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

NameError: name 'spam' is not defined

>>> '2' + 2

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

TypeError: cannot concatenate 'str' and 'int' objects

Iznimke su pogreške tj. odstupanja u radu programa. Mogu biti uzrokovane raznim događajima (pad sistema, nedozvoljeno pisanje po memoriji, dijeljenje s nulom,...) Kada se detektira takva situacija u python programu podiže se iznimka. Iznimka se zatim prosljeđuje pozivatelju funkcije koji ju obrađuje. U slučaju da nema koda za obradu iznimke program se ruši.   
  
try:  
        # blok naredbi koji se pokusa izvrsiti  
except greska:  
        # blok naredbi koji obraduje iznimku greska  
else:  
        # blok naredbi koji se izvrsava ako nije  podignuta niti jedna iznimka

Primjer 10.1. try-except

#!/usr/bin/python

# exception.py

import sys

class maliExcept(Exception):

    'korisnicki definirana exception klasa'

    def \_\_init\_\_(self,duzina,barem):

        self.duzina = duzina

        self.barem = barem

try:

    s = raw\_input('Unesi nesto > ')

    if len(s)<3:

        raise maliExcept(len(s),3)

except EOFError:

    print 'Zasto EOF?'

except maliExcept,x:

    print 'Unos je bio duzine %d, najmanje je trebao biti %d' %(x.duzina,x.barem)

else:

    print 'nije dignuta nije jedna exception'

EOFError – kada korisnik pritisne ctrl+d

Ako želimo izvršiti neki dio koda bez obzira je li došlo do greške ili nije koristimo Try…Finally  
  
  
Primjer 10.2. Try-Finally

#!/usr/bin/python  
# try\_finally.py  
  
try:  
    f = file('tekst.txt')  
    while True:  
        linija = f.readline()  
        if len(linija) == 0:  
            break  
        print linija,  
finally:  
    print 'Ciscenje...'  
    f.close()  
  
$ python try\_finally.py  
Ovo je prva linija tekst.  
ovo je druga linija teksta.  
Ovo je treca linija teksta.  
Ciscenje...

# USPOREDBA DVA PROGRAMSKA JEZIKA(PHP I PYTHON)

Kada se usporedi PHP s Python-om, PHP je skriptni jezik koji se ugrađuje direktno u sadržaj dokumenta. Vrlo je koristan pri manjim web stranicama, no kako se internet razvija aplikacije postaju sve kompleksnije. PHP kod koji se ugrađuje unutar HTML oznaka se prebacuje u predložak odnosno u posebne datoteke koje služe za prikaz, dok je kod poslovne logike posebno prebačen u druge datoteke radi bolje jasnoće i održavanja. Jedan od nedostataka PHP-a je integracija baze podataka direktno u kod, što je dugoročno gledano nedostatak PHP-a, jer je tada baza podataka čvrsto povezana s PHP-ovim određenim funkcijama. PHP-ova sintaksa je slična sintaksi C, varijabla se mora deklarirati sa znakom „$“ i treba biti odmah inicijalizirana, pristup globalnoj varijabli može biti omogućen pomoću ključne riječi „global“.

Python je poznat po kodu koji je čitljiv te kod koji ima jednostavnu sintaksu, koja se može lagano naučiti. Python je bio inspiriran ABC programskim jezikom te s nekoliko značajki koje dolaze od programskog jezika C. Python-ov kod je čitljiv i 38 jednostavan, sintaksa je prilično slična BASIC-ovoj. Njegova je sintaksa i semantika svedena na minimum, dok je standardna biblioteka jako velika. Za razliku od drugih jezika, niti jedan konstrukt u Python-u ne mora imati eksplicitnu oznaku za kraj odnosno oznaka „;“. Python se može proširiti s C ili C++, takva radnja se primjenjuje kada je potrebna vremenski kritička primjena. Najveća prednost je izražajna i čitljiva sintaksa, što znači da se s vrlo kratkim kodom može učiniti mnogo. Python je relativno spor pri izvršavanju u usporedbi s programskim jezikom kao što je C. Python je skriptni jezik koji nije fokusiran samo na web aplikacije, stoga je trebalo neko vrijeme da Python postane značajan za web. Prvi pristup je bio pomoću CGI skripti. Za razliku od PHP-a, Python je od početka bio objektno orijentirani jezik, no nije ograničen na paradigmu objektno orijentiranog jezika, također podržava proceduralno programiranje i neke funkcionalne značajke. U usporedbi s ostalim skriptnim jezicima Python je najjednostavniji skriptni jezik za korištenje, te ima najjednostavniju sintaksu. Python-ova ideja tabulatora umjesto zagrada u kodu možda izgleda sjajno, ali to je ozbiljan nedostatak pri uklanjanju pogrešaka programa. U usporedbi s PHP kodom, vide se razlike, Python-ova sintaksa je lagana za čitati i razumjeti, jer postoji samo mali skup ključnih riječi. Razlikuje se od PHP-a po jednom velikom svojstvu, a to je obavezno uvlačenje blokova unutar koda. To znači da je izvorni kod uvijek pravilno struktuiran, skriptni jezik poput JavaScript, PHP i Perl-a to ne provodi. Kod napisan u Python-u ne treba biti odvojen s tačkom zarezom, jer nova linija koda definira kraj prethodne, no nije zabranjeno. Može se vidjeti iz primjera da se varijabla inicijalizira samo s nazivom te se dodijeli vrijednost, nije potrebno kao u PHP-u dodavati ispred varijable znak „$“.

# ZAKLJUČAK

Python je jednostavan programski jezik koji istovremeno ima snagu i složenost tradicionalnih programskih jezika (C, C++, …) i jednostavnost skriptnih i interpreterskih jezika (Java, Perl, …) . Iako je ponekad identificiran kao čisto skriptni jezik, upotrebljavan je u mnogim velikim projektima kao što je Zope aplikacijski server, Google i Mnet.   
Čitanje dobrog Python programa izgleda gotovo kao čitanje engleskog. Vrlo je lako započeti u njemu programirati, a dovoljno je kompleksan da se u njemu mogu programirati i najteži problemi. Za razliku od većine drugih programskih jezika, ne prisiljava programera da prihvaća jednu programsku paradigmu i razmišlja o samoj metodi programiranja, već da razmišlja o rješavanju problema i onog što zapravo želi učiniti s programom.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 [[7]](#footnote-7)“open source” principima to se uistinu lako postiže. Od jednostavnih početničkih aplikacija do znanstvenoistraživačkih 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.  
Programiranje u Python-u treba biti zabava...

# LITERATURA

## Knjige:

1. Wesley J. Chun: Core Python programming
2. Mark Lutz: Introduction to Programming Python
3. Mark Lutz: Programming Python, 2nd Edition
4. C.H. Swaroop: A Byte of Python

## Linkovi:

1. <https://swaroopch.gitbook.io/byte-of-python/>
2. <https://www.datacamp.com/>
3. <http://shop.oreilly.com/product/9780596000851.do>
4. [https://opentechschool.github.io/python-beginners/en/index.html#](https://opentechschool.github.io/python-beginners/en/index.html)

1. Izvor: <https://en.wikipedia.org/wiki/Monty_Python%27s_Flying_Circus> [↑](#footnote-ref-1)
2. Tk(tkinter) je Pythonov de-facto standardni GUI paket [↑](#footnote-ref-2)
3. Py-je knjižnica za podršku razvoja Python koja sadrži alate i module [↑](#footnote-ref-3)
4. Type-checked: značajka programskog jezika koja određiva kako se varijable stvaraju i njihove tipove prepoznaje prevodilac. [↑](#footnote-ref-4)
5. Echo: je mali Python modul za kreiranje svojstva na koje možete priključiti funkcije povratnog poziva. [↑](#footnote-ref-5)
6. Dir(): funkcija koja vraća sva svojstva i metode navedenog objekta,bez vrijednosti. [↑](#footnote-ref-6)
7. Open-source: je decentralizovani model razvoja softvera koji potiče otvorenu saradnju. [↑](#footnote-ref-7)