

# **Lekcija 6 - Uvod u Python skripting jezik**

# Contents

<b>LearningObject.....</b>	<b>3</b>
L6: Uvod u Python skripting jezik.....	3
UvodnaRazmatranja.....	3
Windows instalacija Python okruženja.....	3
Linux instalacija Python okruženja.....	6
Korišćenje Python skripting jezika.....	8
Ko koristi Python?.....	9
Uvod u Python.....	10
Python upotreba.....	12
Podrška Python programiranju.....	13
Izvršavanje u IDLE okruženju.....	14
Standardni tipovi podataka.....	19
Python brojevi.....	20
Python liste.....	23
Python stringovi.....	27
Python n-torke.....	33
Python prioriteti operatora.....	36
Kontrolne strukture.....	39
Python Break statement.....	44
Python Continue statemen.....	45
Python programiranje.....	48
Tipovi Python varijabli.....	51
Izvršavanje programa.....	52
UNIX izvršni skriptovi.....	55
L6: Uvod u Python skripting jezik.....	58
 <b>Learning Object.....</b>	 <b>59</b>
Uvod u python.....	59
Perl klase i objekti.....	64

# LearningObject

---

## L6: Uvod u Python skripting jezik

---

### Uvod \*

Ovo poglavlje predstavlja uvod u Python skripting jezik.

## UvodnaRazmatranja

---

### Cilj

- Uvodna razmatranja

### Uvodna razmatranja

Suština programiranja svodi se na to da učinite da računar radi ono što vi zadate. To nije najbolja tehnička definicija programiranja, ali je prilično tačna. Pošto savladate Python, znaćete da napišete program, bez obzira na to da li je u pitanju jednostavna igra, kratak pomoćni program ili poslovna aplikacija s kompletnim grafičkim korisničkim interfejsom GUI.

Python je programski jezik opšte namjene koji se često koristi skripting uligama. Obično se definiše kao objektno orijentisani OO jezik. Korisnici često koriste termin“script” umjesto“program” da opišu Python datoteku koda.

## Windows instalacija Python okruženja

---

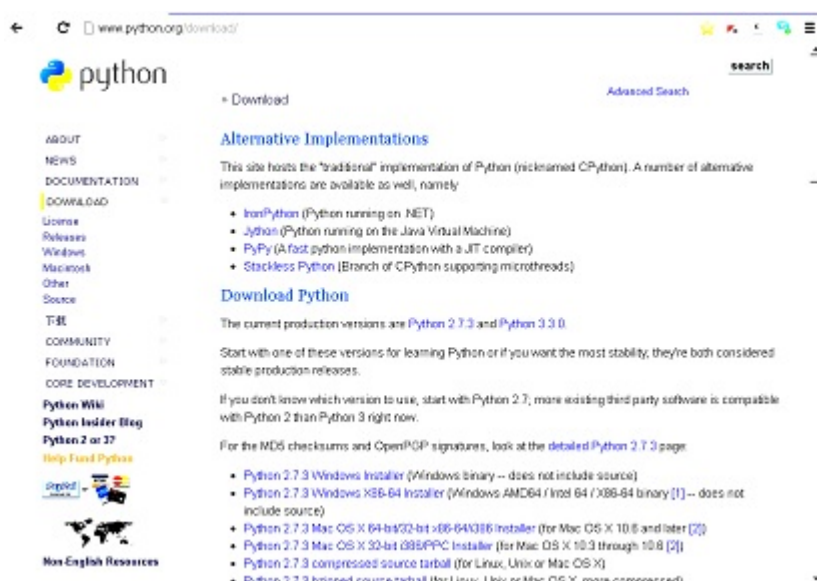
### Cilj

- Postavljanje Pythona na Windows OS

### Windows instalacija Python okruženja

Instalacija Pythona na Windows OS

Sa sajta [www.python.org/download](http://www.python.org/download) preuzeti najnoviju verziju (python-2.7.3.msi), Slika 1. Instalacija je slična instalaciji svakog drugog Windows-baziranog softvera.



Slika 1 Python.org/download sajt

Treba napomenuti da je oko 70% Python preuzimanja pripada Windows korisnicima. Da ne bude zabune, Python je instaliran na većini Linux distribucija.

U interaktivnom načinu rada korisnik ukucava komandu koja se odmah izvršava

U skript način rada, korisnik kreira skup komandi u datoteci ( Python skript ekstenzija je .py) i onda je izvršava.

Provjera da li je Python instaliran:

```
C:\>python -V
```

Python 2.7.3

Korišćenje Python programa sa Windows komandne linije

Prvo treba postaviti PATH variablu.

Procedura

Control Panel à System à Advanced à Environment Variables.

Kliknuti na varijablu PATH u sekciji 'System Variables', izabrati Edit i dodati C:\Python27 na kraju niza.

Drugi način je je ukucavanje komande:

```
set path=%path%;C:\python27
```

na DOS komandnoj liniji.

Dodatak

Download Python

The current production versions are Python 2.7.6 and Python 3.3.5.

Start with one of these versions for learning Python or if you want the most stability; they're both considered stable production releases.

If you don't know which version to use, try Python 3.3. Some existing third-party software is not yet compatible with Python 3; if you need to use such software, you can download Python 2.7.x instead.

For the MD5 checksums and OpenPGP signatures, look at the detailed Python 3.3.5 page:

Python 3.3.5 Windows x86 MSI Installer (Windows binary -- does not include source)

Python 3.3.5 Windows X86-64 MSI Installer (Windows AMD64 / Intel 64 / X86-64 binary [1] -- does not include source)

Python 3.3.5 Mac OS X 64-bit/32-bit x86-64/i386 Installer (for Mac OS X 10.6 and later [2])

Python 3.3.5 Mac OS X 32-bit i386/PPC Installer (for Mac OS X 10.5 and later [2])

Python 3.3.4 compressed source tarball (for Linux, Unix or Mac OS X)

Python 3.3.5 xzipped source tarball (for Linux, Unix or Mac OS X, better compression)

For the MD5 checksums and OpenPGP signatures, look at the detailed Python 2.7.6 page:

Python 2.7.6 Windows Installer (Windows binary -- does not include source)

Python 2.7.6 Windows X86-64 Installer (Windows AMD64 / Intel 64 / X86-64 binary [1] -- does not include source)

Python 2.7.6 Mac OS X 64-bit/32-bit x86-64/i386 Installer (for Mac OS X 10.6 and later [2])

Python 2.7.6 Mac OS X 32-bit i386/PPC Installer (for Mac OS X 10.3 and later [2])

Python 2.7.6 compressed source tarball (for Linux, Unix or Mac OS X)

Python 2.7.6 xzipped source tarball (for Linux, Unix or Mac OS X, better compression)

A comprehensive list of the latest release of all major versions is available if you need source code for an older version of Python.

The following testing versions are available:

Python 3.4.0 release candidate 2

### Alternative Implementations

This site hosts the "traditional" implementation of Python (nicknamed CPython). A number of alternative implementations are available as well, namely

IronPython (Python running on .NET)

Jython (Python running on the Java Virtual Machine)

PyPy (A fast python implementation with a JIT compiler)

Stackless Python (Branch of CPython supporting microthreads)

Other parties have re-packaged CPython. These re-packagings often include more libraries or are specialized for a particular application:

ActiveState ActivePython (commercial and community versions, including scientific computing modules)

pythonxy (Scientific-oriented Python Distribution based on Qt and Spyder)

winpython (WinPython is a portable scientific Python distribution for Windows)

Conceptive Python SDK (targets business, desktop and database applications)

Enthought Canopy (a commercial distribution for scientific computing)

Portable Python (Python and add-on packages configured to run off a portable device)

PyIMSL Studio (a commercial distribution for numerical analysis – free for non-commercial use)

Anaconda Python (a full Python distribution for data management, analysis and visualization of large data sets)

eGenix PyRun (a portable Python runtime, complete with stdlib, frozen into a single executable file)

Information about specific ports, and developer info:

Windows (and DOS)

Macintosh

Other platforms

Source

Python Developer's Guide

Python Issue Tracker

## Linux instalacija Python okruženja

---

### Cilj

- Upoznavanje sa instalacijom Python jezika na Linux OS

### Instalacija Pythona na Linux OS

Instalacija Pythona na Linux OS

Instalacija RedHat Linux OS

Procedura

localhost:~\$ su -

Password: [enter your root password]

```
[root@localhost root]# wget http://python.org/ftp/python/2.3/rpms/redhat-9/
python2.3-2.3-5pydotorg.i386.rpm
```

Resolving python.org... done.

Connecting to python.org[194.109.137.226]:80... connected.

HTTP request sent, awaiting response... 200 OK

Length: 7,495,111 [application/octet-stream]

...

```
[root@localhost root]# rpm -Uvh python2.3-2.3-5pydotorg.i386.rpm
```

Preparing... ##### [100%]

1:python2.3 ##### [100%]

```
[root@localhost root]# python
```

Python 2.2.2 (#1, Feb 24 2003, 19:13:11)

[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-4)] on linux2

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

```

>>> [press Ctrl+D to exit]
[root@localhost root]# python2.3
Python 2.3 (#1, Sep 12 2003, 10:53:56)
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-5)] on linux2
Type "help", "copyright", "credits", or "license" for more information.
>>> [press Ctrl+D to exit]
[root@localhost root]# which python2.3
/usr/bin/python2.3
Provjeriti da li je Python instaliran:
C:\>python -V
Python 2.7.3
Instalacija iz izvornog koda
localhost:~$ su -
Password: [enter your root password]
localhost:~# wget http://www.python.org/ftp/python/2.3/Python-2.3.tgz
Resolving www.python.org... done.
Connecting to www.python.org[194.109.137.226]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8,436,880 [application/x-tar]
...
localhost:~# tar xfz Python-2.3.tgz
localhost:~# cd Python-2.3
localhost:~/Python-2.3# ./configure
checking MACHDEP... linux2
checking EXTRAPLATDIR...
checking for --without-gcc... no
...
localhost:~/Python-2.3# make
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -O3 -Wall -Wstrict-prototypes
-I. -I./Include -DPy_BUILD_CORE -o Modules/python.o Modules/python.c
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -O3 -Wall -Wstrict-prototypes
-I. -I./Include -DPy_BUILD_CORE -o Parser/acceler.o Parser/acceler.c
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -O3 -Wall -Wstrict-prototypes
-I. -I./Include -DPy_BUILD_CORE -o Parser/grammar1.o Parser/grammar1.c
...
localhost:~/Python-2.3# make install
/usr/bin/install -c python /usr/local/bin/python2.3
...

```

```
localhost:~/Python-2.3# exit
```

```
logout
```

```
localhost:~$ which python
```

```
/usr/local/bin/python
```

```
localhost:~$ python
```

```
Python 2.3.1 (#2, Sep 24 2003, 11:39:14)
```

```
[GCC 3.3.2 20030908 (Debian prerelease)] on linux2
```

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

```
>>> [press Ctrl+D to get back to the command prompt]
```

```
localhost:~$
```

Uz Python se isporučuje i integrisano razvojno okruženje pod imenom IDLE (engl. Python Integrated Development Environment). Više informacija se može na dostupno na adresi:

<http://docs.python.org/3/library/idle.html>.

IDLE je kompletno napisano u Python jeziku i Tkinter GUI (<http://en.wikipedia.org/wiki/Tkinter>) alatu i Tcl/Tk funkcijama ).

IDLE se sastoji od skupa alatki koje olakšavaju pisanje programa. Možete ga zamisliti kao program za obradu teksta, s tim što ne služi za tekst nego za programe. Ali, IDLE nije samo mesto gde cete pisati, snimati na disk i menjati svoje programe, vec ima i dva radna režima:

interaktivni način rada

skript način rada

## Korišćenje Python skripting jezika

---

### Cilj

- Zbog čega se koristi Python?

### Korišćenje Python skripting jezika

Razlozi zašto se koristi Python skripting jezik su:

Koristi se za razvoj kvalitetnog softvera

Python kod je čitljiv jezik, može se ponovno koristiti (engl. reusable) i lako se održiva, mnogo bolje nego tradicionalni skriptni jezici. Osim toga, Python ima duboku podršku za više naprednih softverskih mehanizama što su objektno-orijentirano programiranje (OOP).

Poboljšava razvojna produktivnost

Python povećava razvojnu produktivnost koja je mnogo veća nego kod prevođenih ili statičkih tipiziranih jezika jezika (engl. statically typed languages) poput jezika kao što su C, C + +, i Java. Python programski kod je obično jedna trećina do jedne petina veličine ekvivalentnog C + + ili Java koda. Dakle, manje ukucavanja programa, manje vremena za ispravljanje programa. Takođe, Python programi se odmah izvršavaju, bez potrebe dugotrajnog prevođenja i linkovanja čime se još više povećava brzina programiranja.

Programska portabilnost



Većina Python programa se može izvršavati bez ikakvih promjena koda na svim glavnim platformama. Prenosivost Porting Python koda između Linux i Windows OS, na primjer, se svodi na kopiranje skript koda između mašina. Takođe Python nudi više opcija za programiranje prenosivog koda za grafički interfejs, za pristup bazama podataka, web bazirane sisteme i slično.

Python je dostupan na:

Linux and Unix systems

Microsoft Windows i DOS

Mac OS (OS X i Classic)

BeOS, OS/2, VMS, i QNX

Real-time systems (na pr. VxWorks)

Cray superračunar i IBM mainframe

PDA's sa Palm OS, PocketPC i Linux

Cell phones sa Symbian OS i Windows Mobile

Gaming konzole i iPods

Podrška s bibliotekama

Python posjeduje veliku kolekciju unaprijed kreiranih i prenosivih funkcija koje su poznate pod nazivom standardna biblioteka. Ova biblioteka podržava niz programskih zadataka na aplikacionom novou, od „text pattern matching“ do „network scripting“.

Podrška objektnom programiranju

Python je je objektno-orijentisan programski jezik. Njegov model klasa podržava napredne karakteristike kao što si "polymorphism", "operator overloading" i "multiple inheritance". I pored toga, kontekst jezika Python ina jednostavnu sintaksu.

Component integration

Python skriptovi lako komuniciraju sa drugim dijelovima aplikacija koristeći različite integracione mehanizme. Python programski kod može pozvati C i C++ biblioteke, može se pozvati iz C i C++ programa, može se integrisati s Javom i .NET komponentama, može komunicirati preko struktura kao što su COM, može komunicirati s uređajima preko serijskog porta, i mogu komunicirati preko mreže s interfejsima kao što su SOAP, XML-RPC i CORBA.

Zabava

Zbog lakog korišćenja i ugrađenim alatima, Python je postao popularan jezik.

Dakle da li je Python skriptni jezik ili ne? To zavisi o tome koga pitate. U celini, Pojam "skriptni" je vjerovatno najbolje koristi za opisivanje brzog i fleksibilnog način razvoja koji Python podržava, nego neki određeni aplikacioni domen.

## Ko koristi Python?

---

### Cilj

- Korišćenje Python jezika

## Ko koristi Python?

Ko koristi Python?

Zbog toga što je Python je "open source", teško je odrediti koliko korisnika koristi Python. Python je automatski je uključen u Linux distribucije, Macintosh računare, i neke proizvode i hardver. Python se široko koristi već 19 godina, vrlo je stabilan i robustan jezik. Koriste ga individualni korisnici i velike kompanije.

Na primjer, Google intenzivno koristi Python kod pretraživanaj web sistema i koristi:

Python's creator.

The YouTube video sharing service is largely written in Python.

The popular BitTorrent peer-to-peer file sharing system is a Python program.

Google's popular App Engine web development framework uses Python as its application language.

EVE Online, a Massively Multiplayer Online Game (MMOG), makes extensive use of Python.

Maya, a powerful integrated 3D modeling and animation system, provides a Python scripting API.

Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.

Industrial Light and Magic, Pixar, and others use Python in the production of animated movies.

JPMorgan Chase, UBS, Getco, and Citadel apply Python for financial market forecasting.

NASA, Los Alamos, Fermilab, JPL, and others use Python for scientific programming tasks.

iRobot uses Python to develop commercial robotic device

## Uvod u Python

---

### Cilj

- Uvod u Python programski jezik

### Uvod u Python programski jezik

Python je moćan i lako razumljiv jezik koji je smislio Guido van Rossum, a prva verzija je objavljena 1991. godine. Python omogućava da brzo napišete jednostavne projekte. Ali Python se lepo skalira i može se upotrebiti i za pisanje važnih komercijalnih aplikacija.

Jezik je dobio ime po engleskoj komičarskoj trupi Monti Pajton. Uprkos tome što mu je Guido van Rossum nadenuo ime Python po toj trupi, zvanična maskota jezika postala je zmija piton (što je pametna ideja, jer bi bilo prilično teško smestiti lica svih šest britanskih komičara na ikonicu programa). Većina popularnih jezika, kao što su Visual Basic, C# i Java su jezici visokog nivoa i bliži su govornim jezicima nego mašinskom jeziku. Python, sa svojim jasnim i jednostavnim pravilima, još je bliži engleskom govornom jeziku. Pisanje programa na jeziku Python toliko je jednostavno da je nazvano "programiranje brzinom razmišljanja".

Posledica svega toga jeste veća produktivnost profesionalnih programera. Programi na jeziku Python kraci su i pišu se za manje vremena nego programi na mnogim drugim popularnim jezicima. Postoji mnogo programskih jezika. Po čemu je Python tako izuzetan?

Većina popularnih jezika, kao što su Visual Basic, C# i Java su jezici visokog nivoa i bliži su govornim jezicima nego mašinskom jeziku. Python, sa svojim jasnim i jednostavnim pravilima, još je bliži engleskom govornom jeziku. Pisanje programa na jeziku Python toliko je jednostavno da je nazvano "programiranje brzinom razmišljanja". Posledica svega toga jeste veća produktivnost profesionalnih programera. Programi na jeziku Python kraci su i pišu se za manje vremena nego programi na mnogim drugim popularnim jezicima.

Najvažniji cilj svakog programskog jezika jeste da premosti jaz između programe- rovog mozga i računara. Većina popularnih jezika za koje ste možda čuli, kao što su Visual Basic, C# i Java smatra se jezicima visokog nivoa, što znači da su bliži govornim jezicima nego mašinskom jeziku. A oni to zaista i jesu. Ali Python, sa svojim jasnim i jednostavnim pravilima, još je bliži engleskom govornom jeziku.

Pisanje programa na jeziku Python toliko je jednostavno da je nazvano "programiranje brzinom razmišljanja". Jednostavne upotrebe Pythona rezultira u većoj produktivnosti profesionalnih programera. Programi na jeziku Python kraci su i pišu se za manje vremena nego programi na mnogim drugim popularnim jezicima.

Python ima svu moć koju biste i očekivali od savremenog programskog jezika. Kada dodete do kraja ove knjige, bićete u stanju da pišete programe koji rade s grafičkim korisničkim interfejsom, obrađuju datoteke i rade s raznim vrstama struk- tura podataka.

Python je dovoljno moćan da privlači programere širom sveta, i kompanije kao što su Google, IBM, Industrial Light + Magic, Microsoft, NASA, Red Hat, Veri- zon, Xerox i Yahoo!. Python koriste kao alat i profesionalni programeri igara. Kompanije Electronic Arts, 2K Games i Disney Interactive Media Group objavljuju igre u koje je ugrađen Python.

Objektno orijentisano programiranje (OOP) savremen je pristup rješavanju problema pomoću računara, čija je suština intuitivna metoda predstavljanja informacija i akcija u programu. To svakako nije jedini način pisanja programa, ali je za obim- ne projekte često najbolji.

Jezici kao što su C#, Java i Python, objektno su orijentisani. Ali Python pruža i nešto još bolje. U jezicima C# i Java, primena OOP-a nije pitanje izbora. Zbog toga su kratki programi nepotrebno složeni, pa su programeru početniku neophodna prilično opsežna objašnjenja pre nego što bude u stanju da napravi bilo šta vredno pomena.

Python primenjuje drugačiji pristup. U Pythonu, upotreba OOP tehnika nije obavezna. Cela OOP snaga stoji vam na raspolaganju, ali je možete iskoristiti samo kada vam zatreba. Imate jednostavan programčić za koji vam ne treba OOP? Nema problema. Imate opsežan projekat na kojem radi ekipa programera i za koji je potreban OOP? Može i tako. Python pruža i moć i fleksibilnost.

Python se može integrisati s jezicima kao što su C, C++ i Java. Kada radi s Pythonom, programer može iskoristiti rezultate nečeg već napravlje- nog na nekom drugom jeziku. To takode znači da on ili ona može iskoristiti prednosti koje pružaju drugi jezici, kao što je ubrzanje koje C ili C++ mogu da pruže, a da pri tome uživa u lakoci pisanja programa koja je zaštitni znak programiranja na jeziku Python.

Python radi na raznim platformaam, od ručnih računara do Craya. A ako slučajno nemate superračunar u svom podrumu, Python možete koristiti i na mašinama koje rade pod Windowsom, Macintoshem ili Linuxom. A to su samo prve stavke na spisku.

Programi napisani na Pythonu ne zavise od platforme, što znači sledeće: bez ob- zira na operativni sistem pod kojim pišete program, on će raditi pod svakim drugim operativnim sistemom u kojem postoji Python. Ako program napišete na svom PC-ju, kopiju možete poslati e-poštom prijatelju koji ima Linux ili svojoj tetki koja ima Mac i program će raditi (pod uslovom da vaš prijatelj i tetka imaju instaliran Python na svojim računarima

Python je besplatan. Možete ga instalirati na svoj računar a da vas to nikad ne košta ni pare. Ali, Pythonova licenca vam omogućava i mnogo više od toga. Python možete kopirati i menjati. Možete ga čak i dalje prodavati ako želite (ali ne- mojte još davati otkaz na svom tekucem poslu). Prihvatanje ideala otvorenog koda sastavni su deo onoga što Python čini tako popularnim i uspešnim.

## Python upotreba

---

### Cilj

- Šta se može uraditi sa jezikom Python?

### Python upotreba

Šta se može uraditi sa jezikom Python?

Osim toga što se može koristiti kao programski jezik, može se koristiti za različite zadatke kao alat za druge komponente i samostalne programe. Uloga Python jezika kao programskog jezika opšte namjene je praktično neograničena od razvoja veb sajta do razvoja igrice do robotike i upravljanja svemirskim brodovima.

Domeni upotrebe Python jezika su:

Sistemska programiranje

Python je idelan za pisanje prenosivih systemskih administracionih alata i uslu\nnih programa (ili shell tools). Python progarmi mogu da pretražuju datoteke i direktorije, aktiviraju druge programe, podržavaju paralelno procesiranje a procesima ili nitima itd.

Python standardna biblioteka dolazi sa podržava POSIX daje podršku većini OS alata, datoteka i podršku za "socket", "pipes" mehanizme, višestruke niti, regularne izraze, argument komandne linije, itd.

Grafički korisnički interfejs-GUI

Jednostavnost jezika Python i kratko vrijeme razvoja programa baziranih na ovom jeziku je dobra osnova za GUI programiranje.

Internet skripting

Python dolazi sa standardnim Internet modulima koji dozvoljavaju Python programima da izvršavaju širok opseg mrežnihzadataka. Skriptovi komuniciraju sa socket strukturom, izvlače informacije iz forme koja se šalje serveru CGI skriptovima, prenose datoteke pomoću FTP protokola; parsiraju, generišu i analiziraju XML datoteke, šalju, primaju, generišu i parsiraju email poruke.

Integracija komponenta

Integracijom C biblioteke u Python omogućava Python programu da test i pokreće komponente iz biblioteke. Ubacivanjem Python u druge proizvode omogućava onaj prilagođenje (engl. customization) bez potrebe za ponovno prevođenje čitavog programa (engl. recompile).

Programiranje baze podataka

Python podržava interfejs sa Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite i drugim bazama podataka

Brza izrada prototipova (engl. rapid prototyping)

Moguće je inicijalno napraviti prototip sistema uPython jeziku, a onda ubaciti izabrane komponente u prevođene jezike kao što su C ili C++ za isporuku. Za razliku od nekih drugih alata za prototip, Python ne zahtijeva ponovno pisanje programa kada se prototip konvertuje u gotov proizvod

Numeričko i naučno programiranje

NumPy numeriko proširenje Python jezika uključuje napredne alata kao što su izrada interfejsa prema standardnim matematičkim bibliotekama. Integracijom Python jezika sa numeričkim routines kodiranim u prevođenim jezicima povećava brzinu rada.

Igre, slike, serijski portovi, XML, Roboti

To uključuje:

Game programming and multimedia in Python with the pygame system

Serial port communication on Windows, Linux, and more with the PySerial extension

Image processing with PIL, PyOpenGL, Blender, Maya, and others

Robot control programming with the PyRo toolkit

XML parsing with the xml library package, the xmlrpclib module, and third-party extensions

Artificial intelligence programming with neural network simulators and expert system shells

Natural language analysis with the NLTK package

## Podrška Python programiranju

---

### Cilj

- Kako je Python podržan?

### Podrška Python Programiranju

Kao popularan open source sistem, Python ima veliku i aktivnu razvojnu zajednicu koja reaguje na pitanja i problem i razvija poboljšanja brzinom koju mnogi komercijalni softver programeri smatraju izvanrednom (ako ne i stvarno šokantnom). Python programeri koordiniraju on-line rad s izvornom verzijom sistema.

Promjene slijede formalni PEP (engl. Python Enhancement Proposal) protokol i opsežno testiranje sistema. U stvari, modifikovanje Python sistema danas je otprilike kao angažirani kod mijenjanja komercijalnog softvera, daleko od Python ranih dana.

The PSF (engl. Python Software Foundation) je formalna, neprofitna grupa koja organizuje konferencije i bavi pitanjima intelektualnoga vlasništva. Brojni Python konferencije se održavaju širom, O'Reilly je OSCON i PSF je PyCon su najveći.

Dodatak

Python Software Foundation

*The mission of the Python Software Foundation is to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers.*

*from the Mission Statement page*

The Python Software Foundation (PSF) is a non-profit corporation that holds the intellectual property rights behind the Python programming language. PSF manages the open source licensing for Python version 2.1 and later and owns and protects the trademarks associated with Python.

PSF also runs the North American PyCon conference annually, supports other Python conferences around the world, and funds Python related development with grants program and by funding special projects.

## Izvršavanje u IDLE okruženju

---

### Cilj

- Izvršavanje programa u IDLE okruženju

### Izvršavanje u IDLE okruženju

Izvršavanje program u IDLE okruženju

Nako IDLE poziva, otvara se prozor korisnik može da unosi i izvršava Python komande.

Interaktivni rad

Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32

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

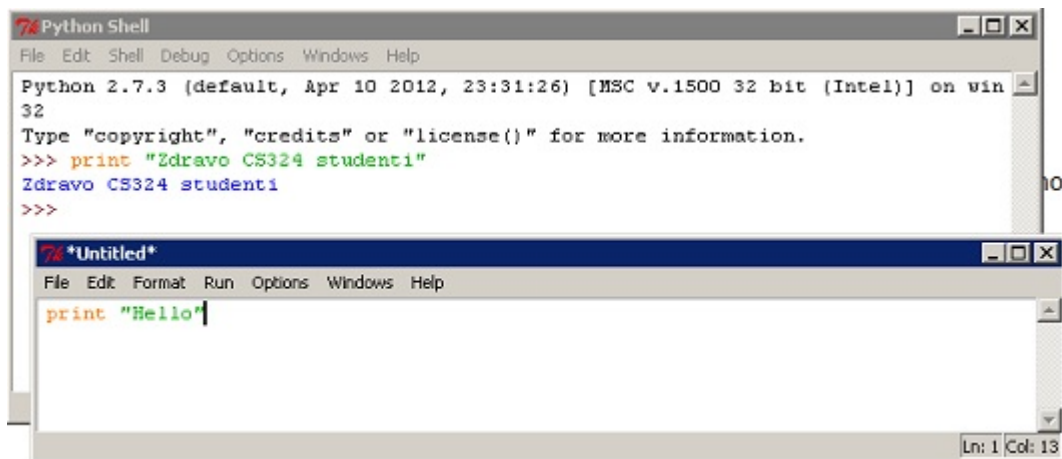
```
>>> print "Zdravo CS324 studenti!"
```

Zdravo CS324 studenti!

```
>>>
```

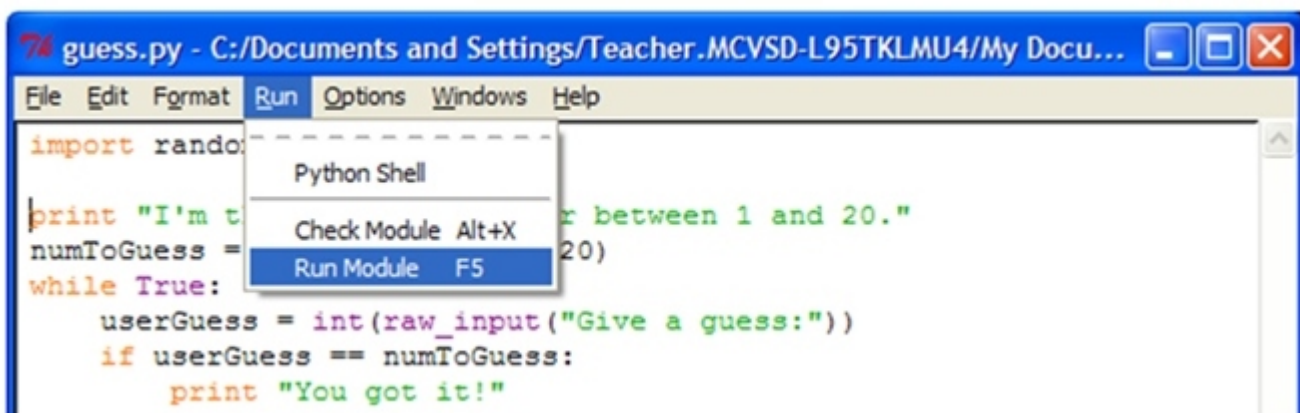
Ovaj se prozor zove REPL (engl. Read-Eval-Print-Loop). Ako korisnik želi napisati Python program, može izabrati sledeće:

File -> New Window za otvaranje prozora u kome može napisati cijeli program, Slika 1.



Slika 1. IDLE okruženje

Za izvršavanje programa, pritisnuti F5 tipku, Slika 2.



Slika 2. Izvršavanje Python programa u IDLE okruženju

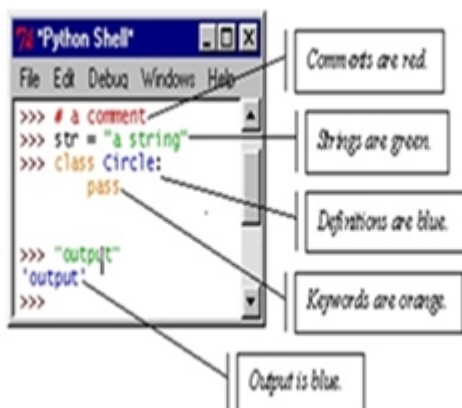
IDLE pomaže Python programerima omogućavajući:

kod sa različitim bojama (Slika 3)

uklanjanje grešaka (engl. debugging)

automatsko uvlačenje pasusa (engl. auto-indent)

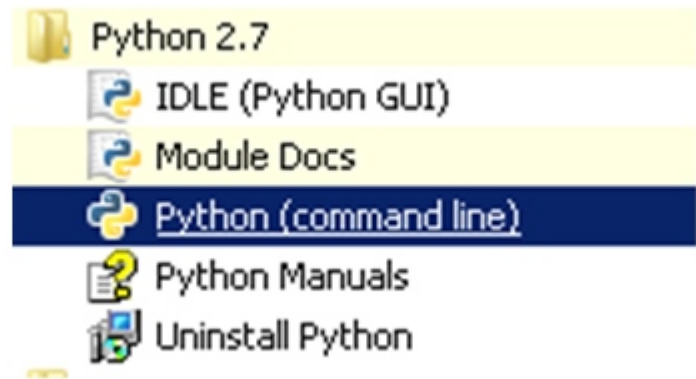
interaktivnu ljusku (engl. shell)



Slika 3 IDLE sa bojama Python koda

Python komandna linija

Slika 4 daje način pozivanja Python komandne linije.



Slika 4 Python komandna linija

Prvi koraci sa interaktivnom ljuskom

Primjer

```
>>> 1 + 1
2
>>> print 'hello world'
hello world
>>> x = 1
>>> y = 2
>>> x + y
3
```

Python interaktivna ljuska ocijenjuje Python izraz, uključujući i osnovne aritmetičke izraze.

Interaktivna ljuska može izvršiti bilo koju Python komandu uključujući i print komandu.

Mogu se zadati i vrijenosti varijablama i one će biti zapamćene sve dok je ljuska otvorena.

Primjer

```
[root@localhost ~]# python
Python 2.4.3 (#1, Jun 18 2012, 08:55:31)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information
>>> x= 128 /4 +512 *2/4
>>> print x
288
```

Slika 5 Primer 1

Primjer



```

Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500]
Type "help", "copyright", "credits" or "license" for more
>>> x= 2048/16 + 32-12*16
>>> print x
-32
>>>

```

## Slika 6 Primer 2

Kod Windows sistema, Python interpreter je obično instaliran kao na C:\Python27

Kod Linux OS, Python interpreter je obično instaliran kao /usr/local/bin/python na onim mašinama na kojima je instaliran. Važno je napomenuti da Python pravi razliku između velikih i malih slova.

## Dodatak

IDLE is an integrated development environment for Python, which has been bundled with the default implementation of the language since 1.5.2b1. It is packaged as an optional part of the Python packaging with many Linux distributions. It is completely written in Python and the Tkinter GUI toolkit (wrapper functions for Tcl/Tk).

An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs offer Intelligent code completion features.

Some IDEs contain a compiler, interpreter, or both, such as Net Beans and Eclipse; others do not, such as SharpDevelop and Lazarus. The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a version control system and various tools are integrated to simplify the construction of a GUI. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development.

## Overview

Integrated development environments are designed to maximize programmer productivity by providing tight-knit components with similar user interfaces. IDEs present a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software. This contrasts with software development using unrelated tools, such as vi, GCC or make.

One aim of the IDE is to reduce the configuration necessary to piece together multiple development utilities, instead providing the same set of capabilities as a cohesive unit. Reducing that setup time can increase developer productivity, in cases where learning to use the IDE is faster than manually integrating all of the individual tools. Tighter integration of all development tasks has the potential to improve overall productivity beyond just helping with setup tasks. For example, code can be continuously parsed while it is being edited, providing instant feedback when syntax errors are introduced. That can speed learning a new programming language and its associated libraries.

Some IDEs are dedicated to a specific programming language, allowing a feature set that most closely matches the programming paradigms of the language. However, there are many multiple-language IDEs, such as Eclipse, ActiveState Komodo, IntelliJ IDEA, Oracle JDeveloper, NetBeans, Codenvy and Microsoft Visual Studio. Xcode, Xojo and Delphi are dedicated to a closed language or set of programming languages.

While most modern IDEs are graphical, text-based IDEs such as Turbo Pascal were in popular use before the widespread availability of windowing systems like Microsoft Windows and the

X Window System (X11). They commonly use function keys or hotkeys to execute frequently used commands or macros.

## History

GNU Emacs, an extensible editor that is commonly used as an IDE on Unix-like systems

IDEs initially became possible when developing via a console or terminal. Early systems could not support one, since programs were prepared using flowcharts, entering programs with punched cards (or paper tape, etc.) before submitting them to a compiler. Dartmouth BASIC was the first language to be created with an IDE (and was also the first to be designed for use while sitting in front of a console or terminal). Its IDE (part of the Dartmouth Time Sharing System) was command-based, and therefore did not look much like the menu-driven, graphical IDEs prevalent today. However it integrated editing, file management, compilation, debugging and execution in a manner consistent with a modern IDE.

Maestro I is a product from Softlab Munich and was the world's first integrated development environment 1975 for software. Maestro I was installed for 22,000 programmers worldwide. Until 1989, 6,000 installations existed in the Federal Republic of Germany. Maestro I was arguably the world leader in this field during the 1970s and 1980s. Today one of the last Maestro I can be found in the Museum of Information Technology at Arlington.

One of the first IDEs with a plug-in concept was Softbench. In 1995 Computerwoche commented that the use of an IDE was not well received by developers since it would fence in their creativity. More recently, James Gosling said: "The number one software development environment for high-end developers these days is still essentially EMACS."

## Topics

### Visual programming

Visual programming is a usage scenario in which an IDE is generally required. Visual IDEs allow users to create new applications by moving programming, building blocks, or code nodes to create flowcharts or structure diagrams that are then compiled or interpreted. These flowcharts often are based on the Unified Modeling Language.

This interface has been popularized with the Lego Mindstorms system, and is being actively pursued by a number of companies wishing to capitalize on the power of custom browsers like those found at Mozilla. KTechlab supports flowcode and is a popular opensource IDE and Simulator for developing software for microcontrollers. Visual programming is also responsible for the power of distributed programming (cf. LabVIEW and EICASLAB software).

An early visual programming system, Max, was modelled after analog synthesizer design and has been used to develop real-time music performance software since the 1980s. Another early example was Prograph, a dataflow-based system originally developed for the Macintosh. The graphical programming environment "Grape" is used to program qfix robot kits.

This approach is also used in specialist software such as Openlab, where the end users want the flexibility of a full programming language, without the traditional learning curve associated with one.

### Language support

Some IDEs support multiple languages, such as GNU Emacs based on C and Emacs Lisp, and Eclipse, IntelliJ IDEA, MyEclipse or NetBeans, all based on Java, or MonoDevelop, based on C#.

Support for alternative languages is often provided by plugins, allowing them to be installed on the same IDE at the same time. For example, Flycheck is a modern on-the-fly syntax checking extension for GNU Emacs 24 with support for 39 languages. Eclipse, and Netbeans have plugins for C/C++, Ada, GNAT (for example AdaGIDE), Perl, Python, Ruby, and PHP, which are selected between automatically based on file extension, environment or project settings.

### Attitudes across different computing platforms

Unix programmers can combine command-line POSIX tools into a complete development environment, capable of developing large programs such as the Linux kernel and its environment. The free software GNU tools (GNU Compiler Collection (GCC), GNU Debugger (gdb), GNU make) are available on many platforms, including Windows. Developers who favor command-line oriented tools can use editors with support for many of the standard Unix and GNU build 1 tools, building an IDE with programs like Emacs or Vim. Data Display Debugger is intended to be an advanced graphical front-end for many text-based debugger standard tools. Some programmers prefer managing make files and their derivatives to the similar code building tools included in a full IDE. For example, most contributors to the PostgreSQL database use make and gdb directly to develop new features. Even when building PostgreSQL for Microsoft Windows using Visual C++, Perl scripts are used as a replacement for make rather than relying on any IDE features. Some Linux IDEs such as Geany attempt to provide a graphical front end to traditional build operations.

On the various Microsoft Windows platforms, command-line tools for development are seldom used. Accordingly, there are many commercial and non-commercial solutions, however each has a different design commonly creating incompatibilities. Most major compiler vendors for Windows still provide free copies of their command-line tools, including Microsoft (Visual C++, Platform SDK, .NET Framework SDK, nmake utility), Embarcadero Technologies (bcc32 compiler, make utility).

IDEs have always been popular on the Apple Macintosh's Mac OS, dating back to Macintosh Programmer's Workshop, Turbo Pascal, THINK Pascal and THINK C environments of the mid-1980s. Currently Mac OS X programmers can choose between native IDEs like Xcode and open-source tools such as Eclipse and Netbeans. ActiveState Komodo is a proprietary multilanguage IDE supported on the Mac OS.

With the advent of cloud computing, some IDEs are available online and run within web browsers; example of this are Online Javascript IDE, Codenvy, Cloud9 IDE, Firebug and Koding.

## Standardni tipovi podataka

---

### Cilj

- Upoznavanje sa standardnim tipovima podataka

### Standardni tipovi podataka

Standardni tipovi podataka

Python has five standard data types:

Numbers

String

List

Tuple

Dictionary

### Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Function	Description
<code>int(x [,base])</code>	Converts x to an integer. base specifies the base if x is a string.
<code>long(x [,base] )</code>	Converts x to a long integer. base specifies the base if x is a string.
<code>float(x)</code>	Converts x to a floating-point number.
<code>complex(real [,imag])</code>	Creates a complex number.
<code>str(x)</code>	Converts object x to a string representation.
<code>repr(x)</code>	Converts object x to an expression string.
<code>eval(str)</code>	Evaluates a string and returns an object.
<code>tuple(s)</code>	Converts s to a tuple.
<code>list(s)</code>	Converts s to a list.
<code>set(s)</code>	Converts s to a set.
<code>dict(d)</code>	Creates a dictionary. d must be a sequence of (key,value) tuples.
<code>frozenset(s)</code>	Converts s to a frozen set.
<code>chr(x)</code>	Converts an integer to a character.
<code>unichr(x)</code>	Converts an integer to a Unicode character.
<code>ord(x)</code>	Converts a single character to its integer value.
<code>hex(x)</code>	Converts an integer to a hexadecimal string.
<code>oct(x)</code>	Converts an integer to an octal string.

Slika 1 Data type conversions

## Python brojevi

---

### Cilj

- Upoznavanje sa Python brojevima

### Python brojevi

Python brojevi

Brojevi se kreiraju kada im se zada vrijednost. Na primjer, `var1 = 1` i `var2 = 10`. Python podržava četiri različita numerička tipa:

`int` (signed integers)

`long` (long integers)

`float` (floating point real values)

`complex` (complex numbers)

Primeri

int	long	float	
10	51924361L	0.0	3.
100	-0x19323L	15.20	45
-786	0122L	-21.9	9.
080	0xDEFABCECBDAECBFBAEI	32.3+e18	.8
-0490	535633629843L	-90.	-.6
-0x260	-052318172735L	-32.54e100	3e
0x69	-4721885298529L	70.2-E12	4.

Slika 1 Primer 1

Sledeći primeri prikazuju razne operacije sa raznim tipovima brojeva.

Primer

```
>>> 2+2
```

```
4
```

```
>>> # This is a comment
```

```
... 2+2
```

```
4
```

```
>>> 2+2 # and a comment on the same line as code
```

```
4
```

```
>>> (50-5*6)/4
```

```
5
```

```
>>> # Integer division returns the floor:
```

```
... 7/3
```

```
2
```

```
>>> 7/-3
```

```
-3
```

Primer

```
>>> 1j * 1j
```

```
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
>>> 1j * 1j
```

Primer

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Dodatak

Python numbers

Number data types store numeric values. They are immutable data types, which means that changing the value of a number data type results in a newly allocated object. Number objects are created when you assign a value to them. For example:

```
var1 = 1
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is:

```
del var1[,var2[,var3[....,varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example:

```
del var
del var_a, var_b
```

Python supports four different numerical types:

int (signed integers): often called just integers or ints, are positive or negative whole numbers with no decimal point.

long (long integers ): or longs, are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.

float (floating point real values) : or floats, represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 ( $2.5e2 = 2.5 \times 10^2 = 250$ ).

complex (complex numbers) : are of the form  $a + bj$ , where  $a$  and  $b$  are floats and  $j$  (or  $j$ ) represents the square root of  $-1$  (which is an imaginary number).  $a$  is the real part of the number, and  $b$  is the imaginary part. Complex numbers are not used much in Python programming.

#### Examples

Here are some examples of numbers (Slika 2):

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.32j
080	0xDEFABCECBDAECBFBAEI	32.3+e18	.876j
-0490	535633629843L	-90.	-.65j
-0x260	-052318172735L	-32.54e100	3e+18j
0x69	-4721885298529L	70.2-E12	4.5j

#### Slika 2 Primer 2

Python allows you to use a lowercase  $L$  with long, but it is recommended that you use only an uppercase  $L$  to avoid confusion with the number  $1$ . Python displays long integers with an uppercase  $L$ .

A complex number consists of an ordered pair of real floating point numbers denoted by  $a + bj$ , where  $a$  is the real part and  $b$  is the imaginary part of the complex number.

#### Number Type Conversion:

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you'll need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

Type `int(x)` to convert  $x$  to a plain integer.

Type `long(x)` to convert  $x$  to a long integer.

Type `float(x)` to convert  $x$  to a floating-point number.

Type `complex(x)` to convert  $x$  to a complex number with real part  $x$  and imaginary part zero.

Type `complex(x, y)` to convert  $x$  and  $y$  to a complex number with real part  $x$  and imaginary part  $y$ .  $x$  and  $y$  are numeric expressions

## Python liste

---

### Cilj

- Upoznavanje sa Python listama

## Python liste

Python liste

Lista sadrži elemente koji su odvojeni zapetama i u dati u uglastim zagradama ([]).

Primjer

```
#!/usr/bin/python
```

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
```

```
tinylist = [123, 'john']
```

```
print list # Prints complete list
```

```
print list[0] # Prints first element of the list
```

```
print list[1:3] # Prints elements starting from 2nd to 4th
```

```
print list[2:] # Prints elements starting from 3rd element
```

```
print tinylist * 2 # Prints list two times
```

```
print list + tinylist # Prints concatenated lists
```

Python ima više složenih tipova podataka koji grupišu druge vrijednosti. Jedan od njih je lista koja može biti napisana kao skup zarezm odvojene vrijednosti između uglastih zagrada. Elementi liste ne moraju da budu istog tipa.

Primer

```
>>> a = ['spam', 'eggs', 100, 1234]
```

```
>>> a
```

```
['spam', 'eggs', 100, 1234]
```

Primer

```
>>> a = ['spam', 'eggs', 100, 1234]
```

```
>>> # Replace some items:
```

```
... a[0:2] = [1, 12]
```

```
>>> a
```

```
[1, 12, 123, 1234]
```

```
>>> # Remove some:
```

```
... a[0:2] = []
```

```
>>> a
```

```
[123, 1234]
```

```
>>> # Insert some:
```

```
... a[1:1] = ['bletch', 'xyzzy']
```

```
>>> a
```

```
[123, 'bletch', 'xyzzy', 1234]
```

```
>>> # Insert (a copy of) itself at the beginning
```

```
>>> a[:0] = a
```

```
>>> a
```



```
[123, 'bletch', 'xyzy', 1234, 123, 'bletch', 'xyzy', 1234]
>>> # Clear the list: replace all items with an empty list
>>> a[:] = []
>>> a
[]
```

Dodatak

### Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Good thing about a list is that items in a list need not all have the same type.

Creating a list is as simple as putting different comma-separated values between square brackets.

For example:

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

Like string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

### Accessing Values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. Following is a simple example:

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];

print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

When the above code is executed, it produces the following result:

```
list1[0]: physics
list2[1:5]: [2, 3, 4, 5]
```

### Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the `append()` method.

Following is a simple example:

```
#!/usr/bin/python

list = ['physics', 'chemistry', 1997, 2000];

print "Value available at index 2 : "
```

```
print list[2];
list[2] = 2001;
print "New value available at index 2 : "
print list[2];
```

Note: append() method is discussed in subsequent section.

When the above code is executed, it produces the following result:

Value available at index 2 :

1997

New value available at index 2 :

2001

### Delete List Elements

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.

Following is a simple example:

```
#!/usr/bin/python
```

```
list1 = ['physics', 'chemistry', 1997, 2000];
```

```
print list1;
```

```
del list1[2];
```

```
print "After deleting value at index 2 : "
```

```
print list1;
```

When the above code is executed, it produces following result:

```
['physics', 'chemistry', 1997, 2000]
```

After deleting value at index 2 :

```
['physics', 'chemistry', 2000]
```

### Basic List Operations

Lists respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
<code>len([1, 2, 3])</code>	3	Length
<code>[1, 2, 3] + [4, 5, 6]</code>	<code>[1, 2, 3, 4, 5, 6]</code>	Concatenation
<code>['Hi!'] * 4</code>	<code>['Hi!', 'Hi!', 'Hi!', 'Hi!']</code>	Repetition
<code>3 in [1, 2, 3]</code>	True	Membership
<code>for x in [1, 2, 3]: print x,</code>	1 2 3	Iteration

Slika 1 Primer

## Python stringovi

---

### Cilj

- Upoznavanje sa Python stringovima

### Python stringovi

Python stringovi

Definisani u Pythonu kao kontinualni niz karaktera.

Primjer

```
str = 'Hello World!'
```

```
print str # Prints complete string
```

```
print str[0] # Prints first character of the string
```

```
print str[2:5] # Prints characters starting from 3rd to 6th
```

```
print str[2:] # Prints string starting from 3rd character
```

```
print str * 2 # Prints string two times
```

```
print str + "TEST" # Prints concatenated string
```

Primjer

```
>>> 'spam eggs'
```

```
'spam eggs'
```

```
>>> 'doesn\'t'
```

```
"doesn't"
```

```
>>> "doesn't"
```

```
"doesn't"
```

```
>>> "'Yes," he said.'
```

```
""Yes," he said.'
```

```
>>> "\"Yes,\" he said."
```

```
""Yes," he said.'
```

```
>>> ""Isn't," she said.'
```

```
""Isn't," she said.'
```

Primer

```
hello = "This is a rather long string containing\n\
```

```
several lines of text just as you would do in C.\n\
```

```
Note that whitespace at the beginning of the line is\
```

```
significant."
```

```
print hello
```

Dva string literala jedan do drugoga se automatski spajaju (negl. concatenation).

Primer

```
>>> 'str' 'ing' # <- This is ok
```

```
'string'
```

```
>>> 'str'. strip() + 'ing' # <- This is ok
```

```
'string'
```

Dodatak

Python strings

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.

Creating strings is as simple as assigning a value to a variable. For example:

```
var1 = 'Hello World!'
```

```
var2 = "Python Programming"
```

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. Following is a simple example:

```
#!/usr/bin/python
```

```
var1 = 'Hello World!'
```

```
var2 = "Python Programming"
```

```
print "var1[0]: ", var1[0]
```

```
print "var2[1:5]: ", var2[1:5]
```

When the above code is executed, it produces the following result:

```
var1[0]: H
```

```
var2[1:5]: ytho
```

### Updating Strings

You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. Following is a simple example:

```
#!/usr/bin/python
```

```
var1 = 'Hello World!'
```

```
print "Updated String :- ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result:

```
Updated String :- Hello Python
```

### Escape Characters

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

An escape character gets interpreted; in a single quoted as well as double quoted strings.

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0-7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0-9, a-f, or A-F

### Slika 1 Primer 1

#### String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Following is a simple example:

```
#!/usr/bin/python
```

```
print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

When the above code is executed, it produces the following result:

My name is Zara and weight is 21 kg!

Here is the list of complete set of symbols which can be used along with %:

# Format Symbol

**%C**

**%S**

**%i**

**%d**

**%u**

**%O**

**%X**

**%X**

## Slika 2 Primer 2

### Triple Quotes:

Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINES, TABs, and any other special characters.

The syntax for triple quotes consists of three consecutive single or double quotes.

```
#!/usr/bin/python
```

```
para_str = """this is a long string that is made up of
several lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed.
NEWLINES within the string, whether explicitly given like
this within the brackets [ \n ], or just a NEWLINE within
the variable assignment will also show up.
"""
```

```
print para_str;
```

When the above code is executed, it produces the following result. Note how every single special character has been converted to its printed form, right down to the last NEWLINE at the end of the string between the "up." and closing triple quotes. Also note that NEWLINES occur either with an explicit carriage return at the end of a line or its escape code (\n):

```
this is a long string that is made up of
several lines and non-printable characters such as
TAB ( ) and they will show up that way when displayed.
NEWLINES within the string, whether explicitly given like
this within the brackets [
], or just a NEWLINE within
the variable assignment will also show up.
```

Raw strings don't treat the backslash as a special character at all. Every character you put into a raw string stays the way you wrote it:

```
#!/usr/bin/python
```

```
print 'C:\\nowhere'
```

When the above code is executed, it produces the following result:

```
C:\nowhere
```

Now let's make use of raw string. We would put expression in r'expression' as follows:

```
#!/usr/bin/python
```

```
print r'C:\\nowhere'
```

When the above code is executed, it produces the following result:

```
C:\\nowhere
```



## Python n-torke

---

### Cilj

- Python n-torke

### Python n-torke

Python n-torke (engl. tuple)

N-torka je još jedan tip sekevenci tipova podaka koje su slične listama. Sastoji se od niza vrijednosti odvojenih zarezima. Za razliku od listi, n-torke su dated u zagradama. Može se reći da su n-torke obično “read-only” liste.

Primer

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2)
tinytuple = (123, 'john')
print tuple # Prints complete list
print tuple[0] # Prints first element of the list
print tuple[1:3] # Prints elements starting from 2nd to 4th
print tuple[2:] # Prints elements starting from 3rd element
print tinytuple * 2 # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

Python rečnik (engl. dictionary)

Python rječnici su heš tabele. Sastoje se od ključeva i i vrednosti.

Primer

```
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print dict['one'] # Prints value for 'one' key
print dict[2] # Prints value for 2 key
print tinydict # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

Dodatak

### Tuples and Sequences

We saw that lists and strings have many common properties, e.g., indexing and slicing operations. They are two examples of *sequence* data types. Since Python is an evolving language, other sequence data types may be added. There is also another standard sequence data type: the tuple.

A tuple consists of a number of values separated by commas, for instance:

```
>>> t = 12345, 54321, 'hello!'
```

```

>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))

```

As you see, on output tuples are always enclosed in parentheses, so that nested tuples are interpreted correctly; they may be input with or without surrounding parentheses, although often parentheses are necessary anyway (if the tuple is part of a larger expression).

Tuples have many uses, e.g., (x, y) coordinate pairs, employee records from a database, etc. Tuples, like strings, are immutable: it is not possible to assign to the individual items of a tuple (you can simulate much of the same effect with slicing and concatenation, though).

A special problem is the construction of tuples containing 0 or 1 items: the syntax has some extra quirks to accommodate these. Empty tuples are constructed by an empty pair of parentheses; a tuple with one item is constructed by following a value with a comma (it is not sufficient to enclose a single value in parentheses). Ugly, but effective. For example:

```

>>> empty = ()
>>> singleton = 'hello', # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)

```

The statement `t = 12345, 54321, 'hello!'` is an example of *tuple packing*: the values 12345, 54321 and 'hello!' are packed together in a tuple. The reverse operation is also possible, e.g.:

```

>>> x, y, z = t

```

This is called, appropriately enough, *tuple unpacking*. Tuple unpacking requires that the list of variables on the left has the same number of elements as the length of the tuple. Note that multiple assignment is really just a combination of tuple packing and tuple unpacking!

Occasionally, the corresponding operation on lists is useful: *list unpacking*. This is supported by enclosing the list of variables in square brackets:

```

>>> a = ['spam', 'eggs', 100, 1234]
>>> [a1, a2, a3, a4] = a

```

## Python Dictionary

A dictionary is mutable and is another container type that can store any number of Python objects, including other container types. Dictionaries consist of pairs (called items) of keys and their corresponding values.

Python dictionaries are also known as associative arrays or hash tables.

The general syntax of a dictionary is as follows:

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

You can create dictionary in the following way as well:

```
dict1 = { 'abc': 456 };
```

```
dict2 = { 'abc': 123, 98.6: 37 };
```

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

### Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

Following is a simple example:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
```

```
print "dict['Name']: ", dict['Name'];
```

```
print "dict['Age']: ", dict['Age'];
```

When the above code is executed, it produces the following result:

```
dict['Name']: Zara
```

```
dict['Age']: 7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
```

```
print "dict['Alice']: ", dict['Alice'];
```

When the above code is executed, it produces the following result:

```
dict['Zara']:
```

```
Traceback (most recent call last):
```

```
File "test.py", line 4, in <module>
```

```
print "dict['Alice']: ", dict['Alice'];
```

```
KeyError: 'Alice'
```

### Updating Dictionary

You can update a dictionary by adding a new entry or item (i.e., a key-value pair), modifying an existing entry, or deleting an existing entry as shown below in the simple example:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
```

```
print "dict['Age']: ", dict['Age'];
print "dict['School']: ", dict['School'];
```

When the above code is executed, it produces the following result:

```
dict['Age']: 8
dict['School']: DPS School
```

### Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

del dict['Name']; # remove entry with key 'Name'
dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age'];
print "dict['School']: ", dict['School'];
```

This will produce the following result. Note an exception raised, this is because after del dict dictionary does not exist anymore:

```
dict['Age']:
Traceback (most recent call last):
File "test.py", line 8, in <module>
print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

## Python prioriteti operatora

---

## Cilj

- Upoznavanja sa Python prioritetima operatora

## Python prioriteti operatora

Python prioriteti operatora (engl. operators precedence)

Slika 1 daje listu Python operatora s njihovim prioritetima.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	<u>C</u> omplement, unary plus and minus; names for the last two are +@ and -@
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular bitwise OR
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= +=  = &= >= <= *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Slika 1 Lista Python operatora s njihovim prioritetima

## Dodatak

### Python Operators Precedence Example

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example, `x = 7 + 3 * 2`; here, `x` is assigned 13, not 20 because operator `*` has higher precedence than `+`, so it first multiplies `3*2` and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Operator	Description
<b>**</b>	Exponentiation (raise to the power)
<b>~ + -</b>	Complement, unary plus and minus (method names for the last two are <code>+=</code> and <code>-=</code> )
<b>* / % //</b>	Multiply, divide, modulo and floor division
<b>+ -</b>	Addition and subtraction
<b>&gt;&gt; &lt;&lt;</b>	Right and left bitwise shift
<b>&amp;</b>	Bitwise 'AND'
<b>^  </b>	Bitwise exclusive 'OR' and regular 'OR'
<b>&lt;= &lt; &gt; &gt;=</b>	Comparison operators
<b>&lt;&gt; == !=</b>	Equality operators
<b>= %= /= //= -= += *= **=</b>	Assignment operators
<b>is is not</b>	Identity operators
<b>in not in</b>	Membership operators
<b>not or and</b>	Logical operators

Slika 2 Primer

Example

Try the following example to understand operator precedence available in Python programming language:

```
#!/usr/bin/python
```

```
a = 20
```

```
b = 10
```

```
c = 15
```

```
d = 5
```

```
e = 0
```

```
e = (a + b) * c / d # ( 30 * 15 ) / 5
```

```
print "Value of (a + b) * c / d is ", e
```

```
e = ((a + b) * c) / d # (30 * 15) / 5
print "Value of ((a + b) * c) / d is ", e

e = (a + b) * (c / d); # (30) * (15/5)
print "Value of (a + b) * (c / d) is ", e

e = a + (b * c) / d; # 20 + (150/5)
print "Value of a + (b * c) / d is ", e
```

When you execute the above program, it produces the following result:

```
Value of (a + b) * c / d is 90
Value of ((a + b) * c) / d is 90
Value of (a + b) * (c / d) is 90
Value of a + (b * c) / d is 50
```

## Kontrolne strukture

---

### Cilj

- Upoznavanje sa Python kontrolnim strukturama

### Kontrolne strukture

If statement

Sintaksa:

if expression:

statement(s)

```
>>> x = int (raw_input("Please enter an integer: "))
```

Please enter an integer: 42

```
>>> if x < 0:
```

```
... x = 0
```

```
... print 'Negative changed to zero'
```

```
... elif x == 0:
```

```
... print 'Zero'
```

```
... elif x == 1:
```

```
... print 'Single'
```

```
... else:
```

```
... print 'More'
```

```
...
```

More

Else statement

Sintaksa:

if expression:

statement(s)

else:

statement(s)

elif statement

Sintaksa:

if expression1:

statement(s)

elif expression2:

statement(s)

elif expression3:

statement(s)

else:

statement(s)

Rezultat

3 - Got a true expression value

100

Good bye!

Nested if...elif...else

Sintaksa:

if expression1:

statement(s)

if expression2:

statement(s)

elif expression3:

statement(s)

else

statement(s)

elif expression4:

statement(s)

else:

statement(s)

while Loop

Sintaksa:

while expression:



statement(s)

for Loop

Sintaksa:

for iterating\_var in sequence:

statements(s)

Iterating by sequence Index

Alternativni način prolaska preko svakog elemenat u iteraciji je dat:

```
fruits = ['banana', 'apple', 'mango']
```

```
for index in range(len(fruits)):
```

```
    print 'Current fruit :', fruits[index]
```

```
print "Good bye!"
```

Dodatak

*if* Statements

Example

```
>>> x = int(raw_input("Please enter an integer: "))
```

```
Please enter an integer: 42
```

```
>>> if x < 0:
```

```
... x = 0
```

```
... print 'Negative changed to zero'
```

```
... elif x == 0:
```

```
... print 'Zero'
```

```
... elif x == 1:
```

```
... print 'Single'
```

```
... else:
```

```
... print 'More'
```

```
...
```

More

*for* Statements

Example

```
>>> # Measure some strings:
```

```
... words = ['cat', 'window', 'defenestrate']
```

```
>>> for w in words:
```

```
...     print w, len(w)
```

```
...
```

```
cat 3
```

```
window 6
```

```
defenestrate 12
```

```
>>> for w in words[:]: # Loop over a slice copy of the entire list.
... if len(w) > 6:
... words.insert(0, w)
...
>>> words
['defenestrate', 'cat', 'window', 'defenestrate']
```

The *range()* Function

The given end point is never part of the generated list; `range(10)` generates a list of 10 values, the legal indices for items of a sequence of length 10. It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the 'step'):

Example

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

*break* and *continue* Statements, and *else* Clauses on Loops

Example

```
>>> for n in range(2, 10):
... for x in range(2, n):
... if n % x == 0:
... print n, 'equals', x, '*', n/x
... break
... else:
... # loop fell through without finding a factor
... print n, 'is a prime number'
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
```

8 equals  $2 * 4$

9 equals  $3 * 3$

The *continue* statement, also borrowed from C, continues with the next iteration of the loop:

```
>>> for num in range(2, 10):
... if num % 2 == 0:
... print "Found an even number", num
... continue
... print "Found a number", num
```

Found an even number 2

Found a number 3

Found an even number 4

Found a number 5

Found an even number 6

Found a number 7

Found an even number 8

Found a number 9

### *pass* Statements

#### Example

The *pass* statement does nothing. It can be used when a statement is required syntactically but the program requires no action. For example:

```
>>> while True:
... pass # Busy-wait for keyboard interrupt (Ctrl+C)
...
```

#### Example

```
x = -6 # Branching
```

```
if x > 0: # If
```

```
print "Positive"
```

```
elif x == 0: # Else if AKA elseif
```

```
print "Zero"
```

```
else: # Else
```

```
print "Negative"
```

```
list1 = [100, 200, 300]
```

```
for i in list1: print i # A for loop
```

```
for i in range(0, 5): print i # A for loop from 0 to 4
```

```
for i in range(5, 0, -1): print i # A for loop from 5 to 1
```

```
for i in range(0, 5, 2): print i # A for loop from 0 to 4, step 2
```

```
list2 = [(1, 1), (2, 4), (3, 9)]
```

```

for x, xsq in list2: print x, xsq # A for loop with a two-tuple as its iterator
l1 = [1, 2]; l2 = ['a', 'b']
for i1, i2 in zip(l1, l2): print i1, i2 # A for loop iterating two lists at once. (The zip function
makes a list of 2-tuples which is then iterated over)
i = 5
while i > 0: # A while loop
i -= 1
list1 = ["cat", "dog", "mouse"]
i = -1 # -1 if not found
for item in list1:
i += 1
if item=="dog":
break # Break; also usable with while loop
print "Index of dog:",i
for i in range(1,6):
if i <= 4:
continue # Continue; also usable with while loop
print "Greater than 4:", i

```

## Python Break statement

---

### Cilj

- Python break statement

### Python break statement

Python break statement

Završava trenutnu petlju i nastavlja na slijedećom izjavom.

Primer

```

if letter == 'h':
break
print 'Current Letter :', letter
var = 10 # Second Example
while var > 0:
print 'Current variable value :', var
var = var -1
if var == 5:
break
print "Good bye!"

```

Dodatak

The Python break Statement:

The break statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both *while* and *for* loops.

Example

```
#!/usr/bin/python
```

```
for letter in 'Python': # First Example
```

```
if letter == 'h':
```

```
    break
```

```
print 'Current Letter :', letter
```

```
var = 10 # Second Example
```

```
while var > 0:
```

```
    print 'Current variable value :', var
```

```
    var = var -1
```

```
if var == 5:
```

```
    break
```

```
print "Good bye!"
```

This will produce the following result:

```
Current Letter : P
```

```
Current Letter : y
```

```
Current Letter : t
```

```
Current variable value : 10
```

```
Current variable value : 9
```

```
Current variable value : 8
```

```
Current variable value : 7
```

```
Current variable value : 6
```

```
Good bye!
```

## Python Continue statemen

---

### Cilj

- Python Continue statement

## Python Continue statement

Continue statement

Vraća kontrolu na početak while petlje.

Primjer

```
for letter in 'Python': # First Example
```

```
if letter == 'h':
```

```
    continue
```

```
print 'Current Letter :', letter
```

```
var = 10 # Second Example
```

```
while var > 0:
```

```
    print 'Current variable value :', var
```

```
    var = var -1
```

```
if var == 5:
```

```
    continue
```

```
print "Good bye!"
```

Pass Statement

To je *null* operacija

Primer

```
#!/usr/bin/python
```

```
for letter in 'Python':
```

```
if letter == 'h':
```

```
    pass
```

```
print 'This is pass block'
```

```
print 'Current Letter :', letter
```

```
print "Good bye!"
```

Dodatak

The continue Statement

The continue statement in Python returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The continue statement can be used in both *while* and *for* loops.

Example

```
#!/usr/bin/python
```

```
for letter in 'Python': # First Example
```

```
if letter == 'h':
```

```
continue
```

```

print 'Current Letter :', letter

var = 10 # Second Example
while var > 0:
    var = var -1
    if var == 5:
        continue
    print 'Current variable value :', var
    print "Good bye!"

```

This will produce following result:

```

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Good bye!

```

### The *pass* Statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The pass statement is a *null* operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

### Example

```

#!/usr/bin/python

for letter in 'Python':
    if letter == 'h':
        pass
    print 'This is pass block'

```

```
print 'Current Letter :', letter
```

```
print "Good bye!"
```

This will produce following result:

```
Current Letter : P
```

```
Current Letter : y
```

```
Current Letter : t
```

```
This is pass block
```

```
Current Letter : h
```

```
Current Letter : o
```

```
Current Letter : n
```

```
Good bye!
```

The preceding code does not execute any statement or code if the value of *letter* is 'h'. The *pass* statement is helpful when you have created a code block but it is no longer required.

You can then remove the statements inside the block but let the block remain with a *pass* statement so that it doesn't interfere with other parts of the code.

## Python programiranje

---

### Cilj

- Upoznavanje sa Python programiranjem

### Python programiranje

Python programiranje

Primer

```
>>> # Fibonacci series:
```

```
... # the sum of two elements defines the next
```

```
... a, b = 0, 1
```

```
>>> while b < 10:
```

```
... print b
```

```
... a, b = b, a+b
```

```
...
```

Rezultat

```
1
```

```
1
```

```
2
```

```
3
```

```
5
```



8

### Python identifikator

Python identifikator je ime koje se koristi za identifikaciju varijabli, funkcija, klasa, modula, ili drugih objekata. Identifikator počinje slovima A do Z i a do z te donjom crtom (\_) koju mogu slijediti karakteri, donje crte i digiti (0 to 9).

Python ne dozvoljava karaktere kao što su @, \$, i % unutar identifikatora. Python razlikuje velika i mala slova (engl. case sensitive). Tako su Fit i fit dva razna identifikatora.

### Python rezervisane reči

Slika1 daje listu Python rezervisanih reči.

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

Slika 1 Python rezervisane riječi

### Redovi i uvlačenje redova (engl. indentation)

Python ne priznaje zagrade za indicaciju blokova koda kod definicije funkcija i klasa ilitoka programa. Blokovi su definisani uvlačenjem koda redova. Broj praznih mjesta je promjenjiv, ali sve izjave u bloku mora biti isto uvučene. Primer bloka:

Primer

```
if True:
    print "True"
else:
    print "False"
```

Kod drugog bloke doći će do greške:

Primer

```

if True:
    print "Answer"
    print "True"
else:
    print "Answer"
    print "False"

```

Izjave sa više linija

Izjave se u Python javljaju obično završavaju sa novim redom (engl. new line). Python međutim dozvoljava nastavak reda korišćenjem (\).

Primjer

```

total = item_one + \
item_two + \
item_three

```

Izjave u okviru zagrada [], {}, ili () ne zahtijevaju posebni karakter za nastavak u novi red.

Primer

```

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

```

Navodni znaci u Python jeziku

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes can be used to span the string across multiple lines. For example, all the following are legal:

Example

```

word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is made up of
multiple lines and sentences."""

```

Komentari u Python jeziku

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the Python interpreter ignores them.

```

#!/usr/bin/python
# First comment
print "Hello, Python!"; # second comment
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.

```

## Tipovi Python varijabli

---

### Cilj

- Tipovi Python varijabli

### Tipovi Python varijabli

Tipovi varijabli u Python jeziku

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals

Zadavanje vrednosti varijablama

The operand to the left of the = operator is the name of the variable, and the operand to the right of the = operator is the value stored in the variable. For example: or characters in these variables.

Primer

```
counter = 100 # An integer assignment
```

```
miles = 1000.0 # A floating point
```

```
name = "John" # A string
```

```
print counter
```

```
print miles
```

```
print name
```

PyScripter okruženje za Python programe

PyScripter predstavlja moćni, nezavisno Python IDE. Kreiran je Delphi okruženju koristeći P4D i SynEdit komponente i Python skripte. Za sada je dostupan samo za Microsoft Windows OS i ima moderni UI.

Dodatak

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

Python variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example:

```
#!/usr/bin/python

counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string

print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles* and *name* variables, respectively. While running this program, this will produce the following result:

```
100
1000.0
John
```

### Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example:

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example:

```
a, b, c = 1, 2, "john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b, and one string object with the value "john" is assigned to the variable c.

## Izvršavanje programa

---

### Cilj

- Izvršavanje Python programa

### Izvršavanje programa

Pogled programera

U najjednostavnijem obliku, Python program je samo tekstualna datoteka koja sadrži Python izjave.

Na primjer, sljedeća datoteka, pod nazivom sc.py, je data kao:

Primjer

```
sc.py
print('hello world')
print(2 ** 100)
```

Izvršavanjem sa DOS komandne linije  
 C:\Python27>python sc.py  
 hello world  
 1267650600228229401496703205376  
 Korišćenje PyScripter IDE softvera

### Python Interpreter

```
*** Remote Interpreter Reinitialized ***
>>>
hello world
1267650600228229401496703205376
>>>
```

Slika 1 Program sc.py

Pogled Python jezika

Byte code compilation

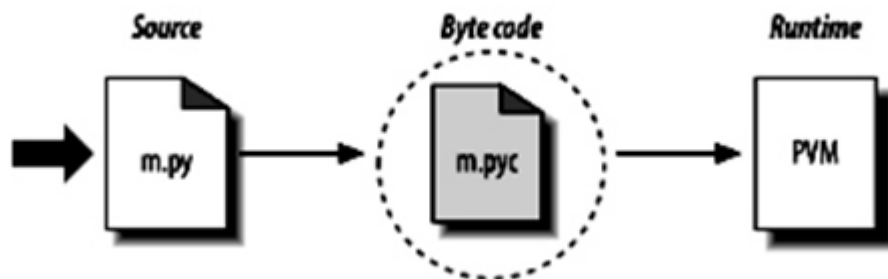
Kada se program izvrši. Python prvo prevede izvorni kod u format poznat kao "byte code". Byte code je reprezentacija izvornog koda u formatu niskog nivoa koja ne zavisi od platforme na kojoj se program izvršava.

Python Virtualna mašina (PVM)

Kada se izvorni kod prevede u bajte codes, prebacuje se na izvršenje u Python virtualnu mašinu PVM.

PVM je samo velika petlja koja prolazi kroz byte code instrukcije, jednu po jednu, da izvrši njihove operacije.

PVM je "run-time" mašina Python jezika, uvijek je prisutna kao dio Python sistema. PVM je tehnički "Python interpreter, Slika2.



Slika 2. Python model izvršavanja programa

Prvi script

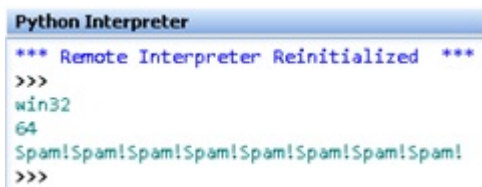
first.py

```
# A first Python script
```

```
import sys # Load a library module
```

```
print (sys.platform) # win32
```

```
print (2 ** 6) # Raise 2 to a power
x = 'Spam!'
print(x * 8) # String repetition
```



```
Python Interpreter
*** Remote Interpreter Reinitialized ***
>>>
win32
64
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
>>>
```

Slika 3 Program first.py

Operacije

Imports a Python module (libraries of additional tools), to fetch the name of the platform

Runs three print function calls, to display the script's results

Uses a variable named x, created when it's assigned, to hold onto a string object

Applies various object operations that we'll begin studying in the next chapter

Izvršavanje sa prompta

```
% python first.py
```

```
win32
```

```
64
```

```
Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

Moguće je usmjeriti izlaz Python skripta u datoteku.

```
% python first.py > saveit.txt
```

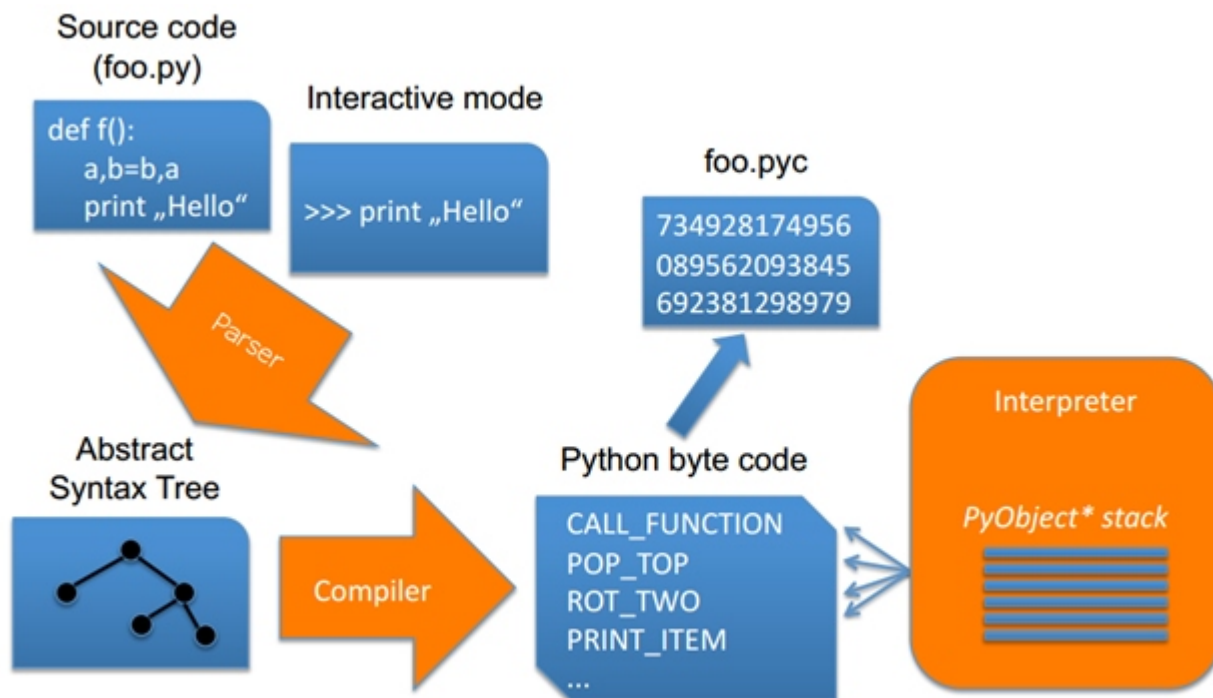
Dodatak

Slika 4 opisuje izvršavanja Python koda.

- **Source code is compiled to some kind of byte code and executed by a virtual machine**
  - CPython: Source code -> Python byte code
    - VM written in C, standard implementation
  - Jython: Source code -> Java byte code
    - JVM as runtime environment
  - IronPython: Source code -> IL (".NET") byte code
    - (".NET") CLR as runtime environment
- **PyPy: Runtime environment written in Python**

Slika 4 Izvršavanja Python koda

Slika 5 ilustruje rad CPython prevodioca i interpretera.



Slika 5 Rad CPython prevodioca i interpretera

## UNIX izvršni skriptovi

### Cilj

- Upoynavanja sa UNIX izvršnim skriptovima

### UNIX izvršni skriptovi

Unix izvršni skriptovi (#!)

U UNIX, Linux ili UNIX-sličnim sistemima moguće je pretvoriti obične Python datoteke u izvršne programe. Takve datoteke se zovu izvršni skriptovi. UNIX izvršni skriptovi su normalne tekst datoteke koje sadrže Python izjave, ali imaju dvije posebne karakteristike:

Skriptovi obično počinju sa redom koji sadrži karaktere `#!` ("hash bang"), a zatim slijedi putanja (engl. path) do Python interpretera na toj mašini.

Obično imaju izvršne privilegije da bi signalizirali OS da ih mogu izvršavati na najvišem nivou.

U Unix sistemu, komanda `chmod +x second.py` izvršava dati zadatak.

Primer

second.py

```
#!/usr/local/bin/python
```

```
print ('The Bright Side ' + 'of Life...')
```

Trik je da se zadrži rezultat na ekranu.

Primer

```
import sys # Load a library module
```

```

print(sys.platform)
print(2 ** 100) # Raise 2 to a power
x = 'Spam!'
print(x * 8) # String repetition
input() # <== ADDED

```

Izlaz

```

C:\Python27>python third.py
win32
1267650600228229401496703205376
Spam! Spam! Spam! Spam! Spam! Spam! Spam! Spam!

```

Slika 1. Third.py

U Windows sistemu, datoteke sa Python programima imaju posebne ikone Windows pretraživaču, Slika 2. Dvostrukim klikovanjem na ikonu Python datoteke, ona se izvršava automatski.

DLLs	11/7/2012 2:54 PM	File Folder	
Doc	11/7/2012 2:54 PM	File Folder	
include	11/7/2012 2:54 PM	File Folder	
Lib	11/7/2012 6:46 PM	File Folder	
libs	11/7/2012 2:54 PM	File Folder	
tcl	11/7/2012 2:54 PM	File Folder	
Tools	11/7/2012 2:54 PM	File Folder	
LICENSE.txt	4/10/2012 11:34...	TXT File	40 KB
NEWS.txt	4/10/2012 11:18...	TXT File	304 KB
python.exe	4/10/2012 11:31...	Application	26 KB
pythonw.exe	4/10/2012 11:31...	Application	27 KB
README.txt	4/10/2012 11:18...	TXT File	54 KB
sc.py	11/7/2012 7:14 PM	Python File	1 KB
third.py	11/7/2012 10:53...	Python File	1 KB
w9xpopen.exe	4/10/2012 11:31...	Application	49 KB

Slika 2 Izvršavanje Python skripta

Funkcija input ()

Funkcija raw\_input() čita svaki tip podataka kao string i program onda procesira string

Funkcija input() koristi funkciju raw\_input() i onda pokušava da konvertuje ulazni podatak u brojnu vrednost koristeći funkciju eval() .

Primer

```

str1 = raw_input("Enter anything:")
print "raw_input =", str1
x = input("Enter a number:")

```



```

print "input =", x
lzlaz
>>>
*** Remote Interpreter Reinitialized ***
>>>
raw_input = CS324
input = 123
>>>

Dodatak

```

### Shebang (Unix)

A "shebang" or "hashbang" character sequence. A shebang (also called a sha-bang, hashbang, pound-bang, hash-exclam, or hash-pling), is the character sequence consisting of the characters number sign and exclamation mark (that is, "#!") at the beginning of a script.

Under Unix-like operating systems, when a script with a shebang is run as a program, the program loader parses the rest of the script's initial line as an interpreter directive; the specified interpreter program is run instead, passing to it as an argument the path that was initially used when attempting to run the script.

For example, if a script is named with the path "path/to/script", and it starts with the following line:

```
#!/bin/sh
```

then the program loader is instructed to run the program `/bin/sh` instead (usually this is the Bourne shell or a compatible shell), passing `path/to/script` as the first argument.

The shebang line is usually ignored by the interpreter because the "#" character is a comment marker in many scripting languages; some language interpreters that do not use the hash mark to begin comments (such as Scheme) still may ignore the shebang line in recognition of its purpose.

### Examples

Some typical shebang lines:

```
#!/bin/sh - Execute the file using sh, the Bourne shell, or a compatible shell
```

```
#!/bin/csh -f - Execute the file using csh, the C shell, or a compatible shell, and suppress the
execution of the user's .cshrc file on startup
```

```
#!/usr/bin/perl -T - Execute using Perl with the option for taint checks
```

Shebang lines may include specific options that are passed to the interpreter. However, implementations vary in the parsing behavior of options; for portability, only one option should be specified (if any) without any embedded whitespace.

### Purpose

Interpreter directives allow scripts and data files to be used as system commands, hiding the details of their implementation from users and other programs, by removing the need to prefix scripts with their interpreter on the command line.

Consider a Bourne shell script that is identified by the path `some/path/to/foo` and that has the following as its initial line:

```
#!/bin/sh -x
```

If the user attempts to run this script with the following command line (specifying "bar" and "baz" as arguments):

```
some/path/to/foo bar baz
```

then the result would be similar to having actually executed the following command line instead:

```
/bin/sh -x some/path/to/foo bar baz
```

If `/bin/sh` specifies the Bourne shell, then the end result is that all of the shell commands in the file `some/path/to/foo` are executed with the positional variables `$1` and `$2` set to `"bar"` and `"baz"`, respectively. Also, because the initial number sign is the character used to introduce comments in the Bourne shell language (and in the languages understood by many other interpreters), the entire shebang line is ignored by the interpreter.

However, it is up to the interpreter to ignore the shebang line; thus, a script consisting of the following two lines simply echos both lines to standard output when run:

```
#!/bin/cat
```

```
Hello world!
```

## L6: Uvod u Python skripting jezik

---

### **Zaključak\***

Nakon studiranja sadržaja ovog poglavlja, studenti će steći početna znanja Python skripting jezika.

# Learning Object

---

## Uvod u python

---

### Cilj

- Upoznavanje sa osnovama Python-a

### Uvod u python

Python REPL se pokreće komandom 'python'. U repl-u se mogu pisati izrazi, definisati funkcije, klase ili pozivati i instancirati druge funkcije i klase. Komentari u Python-u počinju znakom # i traju do kraja linije.

Brojevi:

```
>> 2 + 2
```

```
4
```

```
>> 21/4
```

```
5.25
```

```
>> 3**2
```

```
9
```

Promenljive:

Python nema posebnu sintaksu za deklaraciju promenljivih - promenljiva se kreira kada joj se prvi put dodeli vrednost.

```
>> a = 5
```

```
>> a * 2
```

```
10
```

Stringovi:

```
>> "double quoted"
```

```
'double quoted'
```

```
>> 'single quoted'
```

```
'single quoted'
```

```
>> len("duzina stringa")
```

```
14
```

# konkatencija stringova (vraca novi string)

```
>> str = "Hello "
```

```
>> str = str + "World!"
```

```
>> str
"Hello world!"

# indeksiranje stringova
>> "Hello"[0]
'H'

# slicing - vraca string od prvog indeksa (inkluzivno) do drugog (ekskluzivno)
>> "Hello"[1:5]
'ello'

# Ukoliko se drugi indeks ne definiše, slice-uje se do kraja stringa
>> "Hello"[1:]
'ello'

# Ukoliko se prvi indeks ne definiše, slice-uje se od početka stringa
>> "Hello"[:3]
'Hel'

# Moguce je koristiti i negativne indekse:
>> "Hello"[1:-1]
'ell'
```

#### Interpolacija stringova

Ubacivanje vrednosti izraza u stringove se radi operatorom '%'.

```
>> a = 5
>> b = 4
>> "Zbir %d i %d je %d" % (a, b, a + b)
'Zbir 5 i 4 je 9'
```

Karakter '%' u stringu se menja vrednošću iz n-torke desno od operatora %. Posle karaktera % se nalazi specifikator tipa, u ovom slučaju %d označava da je u pitanju celobrojna vrednost. %s bi označila da je u pitanju string.

#### Pretvaranje vrednosti u stringove

Pretvaranje vrednosti u stringove se radi pozivanjem funkcije str().

```
>> str(5)
'5'
>> str(True)
'True'
```

Ekvivalentne funkcije postoje za pretvaranje stringova u druge tipove:

```
>> float("5")
5.0
```

```
>> int("5")
```

```
>> 5
```

```
>> bool("True")
```

```
True
```

### Nizovi

Nizovi se kreiraju literalom [], kao u JavaScript-u i Ruby-u. Nizovi mogu sadržati elemente bilo kog tipa, uključujući i druge nizove.

```
>> [1, 2, "3", [4, 5]]
```

```
[1, 2, "3", [4, 5]]
```

Sintaksa za indeksiranje i slicing je identična kao za stringove:

```
>> a = [1,2,3,4,5]
```

```
>> a[1:3]
```

```
[2,3]
```

```
>> a[:3]
```

```
[1,2,3]
```

```
>> a[3:]
```

```
[4,5]
```

```
# konkatencija nizova
```

```
>> [1,2,3] + [4,5]
```

```
[1, 2, 3, 4, 5]
```

```
# ponavljanje nizova
```

```
>> [1,2,3] * 2
```

```
[1, 2, 3, 1, 2, 3]
```

```
# duzina niza
```

```
>> len([1,2,3])
```

```
3
```

### n-torke

n-torke su kolekcije koje sadrže fiksni broj elemenata. U pythonu se kreiraju operatorom "(", sa ili bez zagrada.

```
>> 2, 3
```

```
(2, 3)
```

```
>> (3, "a")
```

```
(3, 'a')
```

# n-torke se mogu koristiti za dodeljivanje vrednosti:

```
>> a, b, c = 1, 2, 3
```

```
>> a
```

```
1
```

```
>> b, c
```

```
(2, 3)
```

nizovi se mogu otpakovati na sličan način, pod uslovom da je broj elemenata niza jednak broju promenljivih:

```
>> a, b, c = [1, 2, 3]
```

```
>> a, b, c = [1, 2, 3, 4]
```

```
ValueError: too many values to unpack (expected 3)
```

Boolean vrednosti

promenljive boolean tipa mogu imati samo dve vrednosti: True i False.

```
>> a = True
```

```
>> a == True
```

```
True
```

```
>> a == False
```

```
>> False
```

None vrednost

None predstavlja vrednost analognu null u JavaScript-u i nil u Ruby-u.

Python operatori

Python ima sve poznate relacije, aritmetičke, assignment i logičke operatore. Do sada su korišćeni \*, +, - i ==. Ostale je moguće naći u dokumentaciji na referenciranoj na kraju dokumenta.

## Control flow

Svaki početak novog bloka u python-u se započinje karakterom ":", novom linijom i povećanom indentacijom.

Primer if-a:

```
>>> x = int(raw_input("Please enter an integer: "))
```

```
Please enter an integer: 42
```

```
>>> if x < 0:
```

```
... x = 0
```

```
... print('Negative changed to zero')
```

```
... elif x == 0:
```

```
... print 'Zero'
```

```
... elif x == 1:
```

```
... print('Single')
```

```
... else:
... print('More')
```

```
...
```

More

For petlja

```
>> array = [1, 2, 3, 4, 5]
```

```
>> for item in array:
```

```
... print(item)
```

```
...
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

While petlja

```
>> array = [1,2,3]
```

```
>> i = 0
```

```
>> while i < len(array):
```

```
... print(array[i])
```

```
... i = i + 1
```

```
...
```

```
1
```

```
2
```

```
3
```

Definisanje funkcija

Za definisanje funkcija se koristi keyword def. Domen funkcije je blok koji sledi ispod zaglavlja funkcije (blok mora biti uvučen):

```
def add(a, b):
```

```
    return a + b
```

```
add(2, 3) # vraća 5
```

# funkcije ce baciti exception ukoliko ne dobiju definisan broj argumenata:

```
>> add(2)
```

```
TypeError: add() missing 1 required positional argument: 'b'
```

```
>> add(1,2,3)
```

```
TypeError: add() takes 2 positional arguments but 3 were given
```

Default argumenti

Funkcije u pythonu mogu imati default argumente.

```
def foo(a, b = 5):
```

```
    return a, b
```

```
>> foo(1, 2)
```

```
(1, 2)
```

```
>> foo(1)
```

```
(1, 5)
```

## Perl klase i objekti

---

### Perl - Klase i objekti

1) Napisati funkciju koja prima tri argumenta. Prva dva su brojevi, a treći je string koji predstavlja operaciju. Ukoliko je treći argument "+", funkcija vraća zbir, ukoliko je "\*", vraća proizvod prva dva argumenta. Ukoliko treći argument nije prosleđen, njegova podrazumevana vrednost je "+". Pokazati da funkcija radi sa nekoliko primera (poziva).

2) Napisati sledeće funkcije koje vrše operacije nad jednim nizom:

- **first** - vraća prvi element niza
- **last** - vraća poslednji element niza
- **tail** - vraća sve elemente niza osim prvog
- **init** - vraća sve elemente niza osim poslednjeg

Pokazati da funkcija radi sa nekoliko primera.