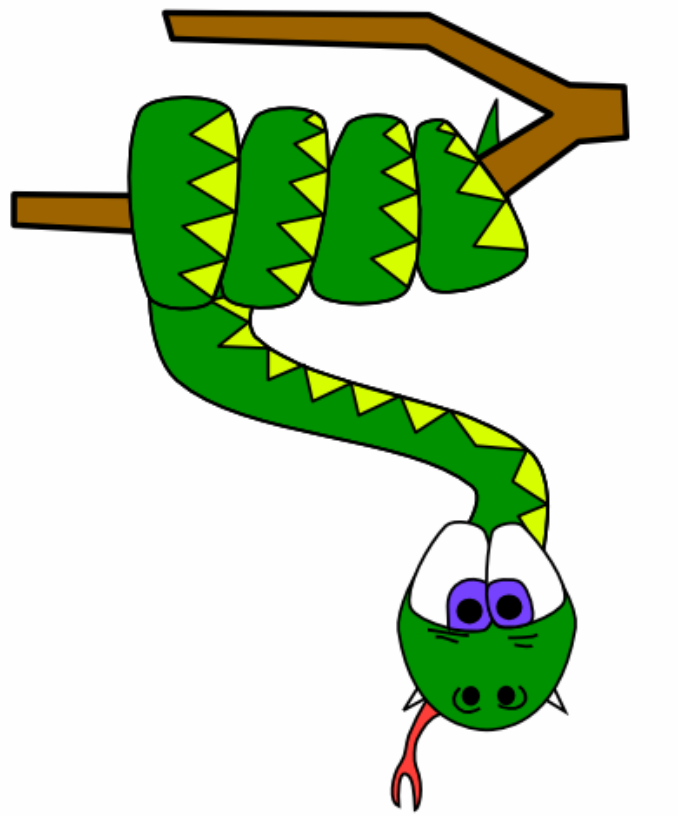
UNIVERZITET U NOVOM SADU
TEHNOLOŠKI FAKULTET

PROGRAMIRANJE I PRIMENA RAČUNARA

PYTHON PROGRAMSKI JEZIK



OKTOBAR, 2004

SADRŽAJ

UVODNA REČ

Ovaj tutorial namenjen je studentima Prehrambenih i Farmaceutskih smerova na Tehnološkom fakultetu u Novom Sadu i ima svrhu da im pomogne u sticanju osnovnih znanja iz programiranja u Python programskom jeziku koje bi sami kasnije primenili na struku. Tutorial je sastavljen na osnovu raznih tutorijala, prevedenih, obrađenih i obogaćenih novim primerima, objašnjenjima i komentarima. Nadamo se da smo uspeli koliko toliko, da sastavimo materijal koji će u znatnoj meri studentima olakšati rad i učenje. Više o praktičnoj primeni Pythona studenti će imati prilike da saznaju na vežbama i časovima.

Želimo Vam mnogo uspeha u Pythonisanju. Ukoliko imate primedbe ili sugestije možete ih poslati na e-mail adresu : racunari@tehnol.ns.ac.yu

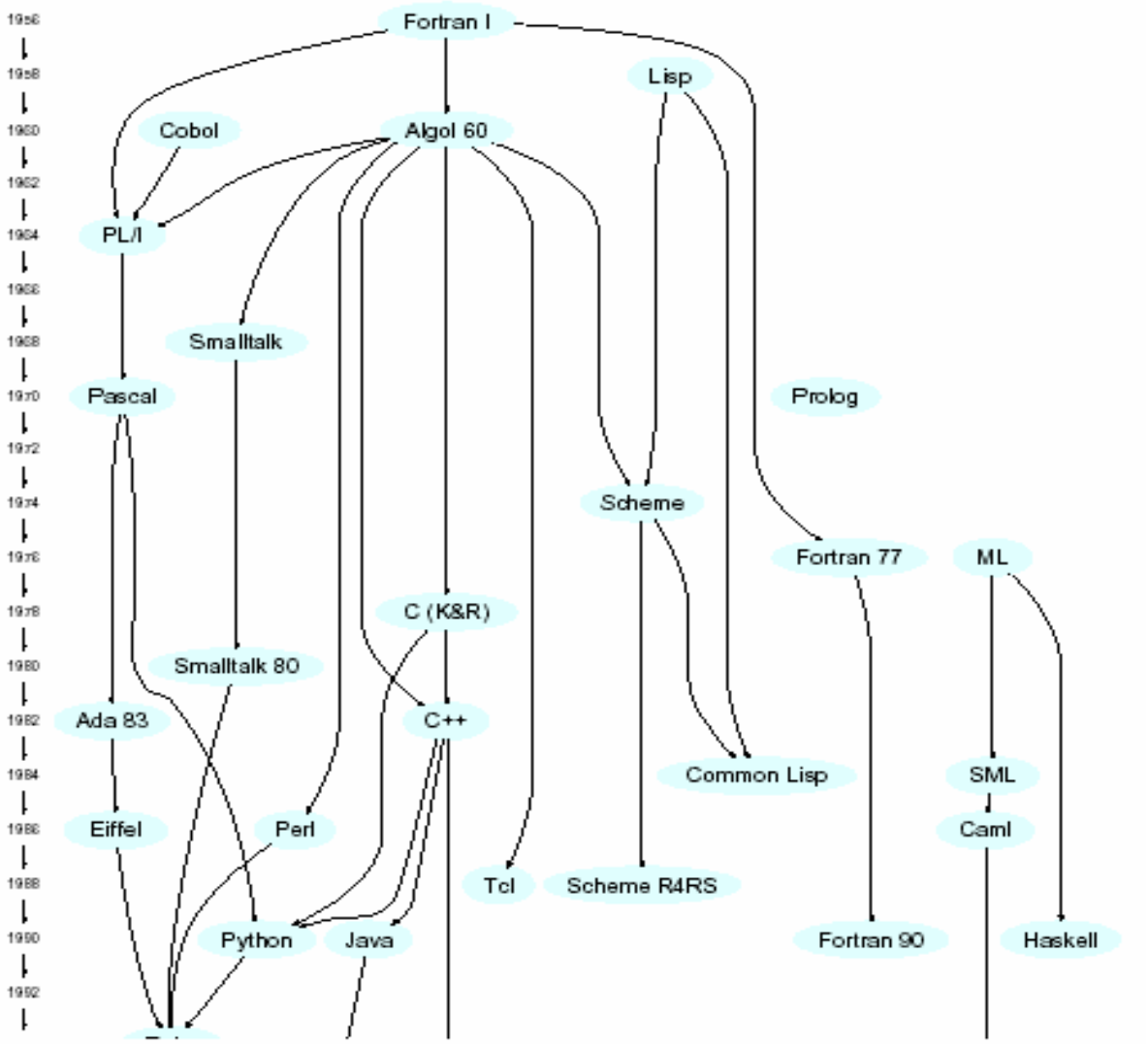
Autori : Vesna Vasić i Jelena Marković

Savetodavac i recezent: Prof. Dr Radovan Omorjan

Sa predmeta: Programiranje i primena računara

1. Uvod u Python

Mnogi ljudi bi hteli započeti programirati, ali jednostavno ne znaju kako. QBasic je bio dobro rešenje pre 10 godina. Danas će vam svi verovatno preporučiti C, C++, Javu, Visual Basic... Loša stvar je, da ni jedan od tih jezika nije lako naučiti bez prethodnog znanja programiranja. Pored toga treba napomenuti da se na raznim inženjerskim fakultetima i univerzitetima izučava široka paleta programskih jezika od kojih su najpopularniji C, C++, Fortran, Basic, Visual Basic, Pascal itd. Takođe, ne postoji konsenzus koji je programski jezik najpogodniji, iako postoji opšta saglasnost da moderan inženjer mora da poznaje više programskih jezika. Evo male šeme istorijskog razvoja programskih jezika:



Koji programski jezik onda izabrati? Ako ste genijalac/ka i imate smisla za programiranje, C++ je verovatno dobro rješenje. Ako ne znate programirati, onda je idealno rešenje Python. Zašto Python? Prvenstveno zato jer je:
* besplatan
* lagan za naučiti
* moćan
* dobro dokumentovan

- Python je slobodan softver otvorenog koda (čitaj besplatan) čiji je razvoj veoma intenzivan i može se besplatno skinuti (download-ovati) sa Interneta (www.python.org). Pored toga postoji za više platformi - Windows, Unix, Linux, Mac OS itd. Qbasic je MSDOS program koji je takođe besplatan ali se više ne razvija. Razvijaju se njegovih komercijalni naslednici - VB, VBA, VBS itd.)

- Izuzetno je lak za korisnika. Autori Pythona su se vodili idejom da se pri rešavanju problema koncentracija posveti samom problemu a ne finesama programskog koda. Postoji mišljenje da je jedan od najlakših programskih jezika za korišćenje (lakši i od Qbasic-a). Samim tim i vreme potrebno za savladavanje osnova Python-a je dosta kratko.

- Pored toga je i izuzetno moćan. Python se koristi u mnogim sferama primene računara: od zamene za kalkulator pa do kompleksnih korisničkih programa, stvaranja web prezentacija itd. Postoji ogromna biblioteka dodataka za Python (besplatni ili komercijalni).

- Iako je sam Python besplatan on se koristi i u komercijalne svrhe za najrazličitije namene. Koriste ga mnogi pojedinci i firme. Navešćemo neke primere: Sim42 je softver za simulačiju hemijskih procesnih postrojenja koji je napisan u Python-u, Biopython je besplatni dodatak za molekularnu biologiju, Bioconductor je besplatan dodatak za bioinformatiku itd. Kao potvrda popularnosti je i to da je on rangiran među prvih deset najpopularnijih i najkorišćenijih programskih jezika u svetu (www.tiobe.com : Python - 8 mesto, avgust 2004)

Python je izvrstan kao početni jezik, a ako ga nastavite učiti, shvatićete kako je moćan.
Pa, počnimo...

Python je objektno-orjentisani jezik. Vama verovatno to sada apsolutno ništa ne znači. To ćemo objasniti u nekom drugom delu.

2. Osnovne stvari

Pokretanje Python-a

Prvo što treba da uradite je da downloadujete Python sa http://http://www.python.org. Velik je oko 7MB, ali se download isplati. Zavisno koji operativni sistem imate (pretpostavljamo da imate Windowse - uz skoro sve Linux distribucije dolazi dobro iskonfigurisani Python), pokrenite instalaciju i instalirajte Python.

Nakon toga, možete pokrenuti Python kliknuvši na **"Start"->"Programs"->"Python 2.3"->"IDLE (Python GUI).** Trebalo bi da dobijete ovakvu poruku:

```
Python 2.3.4 (#53, May 25 2004, 21:17:02) [MSC v.1200 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    **********************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    **********************************************************************

IDLE 1.0.3
>>>
```

>>> (tri strelice) znače da je Python spreman za unos naredbe. Python je interpreterski jezik, što znači da nije moguće da napravite .EXE od vašeg programa, nego program morate pokretati pomoću interpretera.(NOTE: Zapravo, .exe datoteka se može napraviti - postoji jedan program koji pretvara .py u .exe.URL: py2exe)

Naredba print i input, varijable i stringovi

Probajte napisati **print "Python rules!"**. Program bi trebao ispisati **Python rules** na ekranu. Primetite da program odmah izvršava vašu naredbu. Ako prvo želite napisati program, pa onda ga pokrenuti, odaberite **"File"->"New window"** i tu napišite program. Kada završite, spremite program (**"File"->"Save"**) i odaberite **"Edit"->"Run Script"** i program će se pokrenuti.

Postoji tradicija da se pri pisanju uputstva uvek prvo navede program koji ispisuje "Zdravo, svete!" na ekran. U Python-u , ovo je zaista lako :

```
print "Hello, world!"
```

Ovo je praktično kao recept koji govori kompjuteru šta da radi : Da ispiše "Hello, world!". Ništa lakše. Šta ako želimo da uradi više stvari?

```
print "Hello, world!"
print "Goodbye, world!"
```

Nije mnogo teže, zar ne? I ne baš zanimljivo... Mogli bi nešto da uradimo sa sastojcima, baš kao pri pravljenju salate. Pa – koje sastojke imamo? Kao prvo, imamo nizove teksta kao "Hello, world!", ali takođe imamo i brojeve.Python jako dobro služi i kao kalkulator. U prozoru **"Python shell"** upišite neki matematički izraz, npr. **2+2**, i Python će ispisati rezultat. Možete koristiti zagrade, imaginarne brojeve (šta god to bilo), itd.

```
>>> 2+2

4
```

Dobro, počnimo programirati. Naučili smo šta radi naredba print - ispisuje nešto na ekranu. Možemo takođe napisati **print 2+2**

```
>>> print 2+2

4
```

pogledajmo malo drugačiji primer:

```
>>> print '2+2'

2+2
```

Zapažate razliku? Python tretira niz karaktera između navodnika kao string tj. kao tekst, pa se razlikuje rezultat u predhodna dva primera.

Recimo da želimo da kompjuter izračuna površinu pravougaonika za nas. Tada bi mu mogli zadati ovakav mali recept:

```
# Povrsina pravougaonika

# Sastojci:

sirina = 20
visina = 30

# Uputstva:

povrsina = sirina*visina
print povrsina
```

Verovatno možete da vidite sličnosti sa receptom za salatu. Ali kako to radi? Pre svega, linije koje počinju sa # zovu se komentari (recimo u Pascal-u je to {tekst} -prim.prev.) i njih kompjuter ignoriše. Ipak, ubacivanje malih objašnjenja kao što su ova može biti olakšavajuća okolnost za ljude koji čitaju kod vašeg programa. Sada, redovi koji izgledaju kao **nešto = merna jedinica** zovu se dodele. U slučaju sirina = 20 govorimo kompjuteru da širina treba biti 20 . Šta to znači "širina je 20"? To znači da je stvorena promenljiva sa imenom "sirina" (ili ako već postoji, promenjena je njena vrednost) i data joj je vrednost 20. Tako, kada kasnije koristimo tu promenljivu, kompjuter zna njenu vrednost. Zato,

```
sirina*visina
```

Je u stvari isto kao

```
20*30
```

Čijim izračunavanjem dobijamo 600, što se sada dodeljuje promenljivoj po imenu "povrsina". Poslednja linija programa ispisuje vrednost promenljive "povrsina", pa je ono što vidite kada pokrenete program samo

600

Napomena : U nekim jezicima morate kompjuteru da kažete koje su vam promenljive potrebne na samom početku programa (kao spisak sastojaka za salatu) – Python je dovoljno pametan da ih sam shvati idući redom. U Pythonu se ne morate brinuti o deklarisanju tipa varijable (u mnogim drugim programskim jezicima se prilikom dodeljivanja imena promenjivoj mora voditi računa o deklaraciji), Python sam određuje tip varijable, za razliku od C++-a gde prvo svaku varijablu morate deklarisati (zajedno s njenim tipom - integer, double, char, ...).

OK. Sada možete da izvedete jednostavne, ali i veoma složene proračune. Na primer, možda želite da program izračuna površinu kruga umesto pravougaonika :

```
>>> poluprecnik=30

>>> print poluprecnik*poluprecnik*3.14

2826.0
```

Kako bilo, ovo nije mnogo zanimljivije nego program za pravougaonik, nekeko je nepromenljivo. Šta ako krug koji nama treba ima poluprečnik 31? Kako bi kompjuter to znao?. Treba nam podrška, ili INPUT. Sada kada znamo kako ispisivati na ekranu, vreme je da naučimo kako uzimati informacije od korisnika. Za to koristimo naredbu **input**. Kako kompjuter da sazna poluprečnik kruga? I njemu treba ulazna vrednost... Šta mi možemo da mu kažemo to je da pita korisnika koliki je poluprečnik.

```
poluprecnik = input("Koliki je poluprecnik?")
print poluprecnik*poluprecnik*3.14
```

Rezultat programa izgleda ovako:

```
Koliki je poluprečnik?31

3017.54
```

Sad stvari postaju zapetljane... ulaz je nešto što zovemo funkcija. (Uskoro ćete naučiti da stvarate svoje. Input je funkcija koja je ugrađena u Python programski jezik.).
Jednostavno pišući:

```
Input
```

nećemo postići mnogo... Morate da stavite par zagrada na kraju. Tako, input() bi radilo – jednostavno bi čekalo da korisnik unese poluprečnik. Verzija odozgo je možda lakša za korišćenje, jer ipak prvo ispisuje pitanje. Kada stavimo nešto kao niz teksta "Koliki je poluprečnik?" u zagradu funkcije to se zove dodela parametra funkciji. Stvar (ili stvari) u zagradi su parametri. U ovom slučaju dodajemo pitanje kao parametar da input zna šta da ispiše pre nego što dobije odgovor od korisnika.
Ali kako odgovor dospeva do promenljive poluprečnik? Funkcija input, kada je pozvana, vraća vrednost (kao i mnoge druge funkcije). Ne morate da koristite ovu vrednost, ali u našem slučaju, želimo tako. Zato, sledeće dve izjave imaju veoma različito značenje :

```
nesto = input
vrednost = input()
```

nešto sada ima samu funkciju unosa (pa može biti korisćeno kao nešto("koliko imas godina ?"); ovo se zove poziv dinamične funkcije) dok vrednost sadrži ono što je korisnik uneo..

Sledeći program pita korisnika koliko ima godina, i nakon toga ispisuje rezultat na ekranu:

```
godine = input('Koliko imate godina? ')
print godine
```

Prvo se varijabli **godine** pridružuje korisnikov unos (unos traje dok korisnik ne pritisne ENTER), a onda se vrednost varijable ispisuje.

Probajte - otvorite IDLE (Python GUI), odaberite **File->New window**, i tamo kopirajte ovaj program. Snimite ga pod nekim imenom i stisnite F5 da bi ga pokrenuli. Divite se svom prvom programerskom ostvarenju! :)

Umesto broja godina, možemo pitati korisnika da upiše bilo šta i da se to onda ispiše na ekranu. Kod je isti, osim jedne male razlike u naredbi unosa. Naredba više nije **input**, već **raw_input**, pa kod izgleda ovako:

```
tekst = raw_input('Unesite neki tekst: ')
print tekst
```

Znači, za brojeve koristimo naredbu **input**, a za stringove koristimo **raw_input**.

Važna napomena: Pzthon ima dvadeset i osam ključnih reći koje ne bi trebali da koristite za imena promenjivih, jer ih on tumači kao naredbe, a to su:

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

Pogledajmo sledeći primer:

```
a = 2
b = 4
print a + b
```

Unesete li ovaj program i pokrenete li ga, na ekranu ćete dobiti rezultat 6. Zašto? a i b su (kao što već znamo) varijable, i to brojčane, kojima su dodijeljene vrednosti 2 i 4. Python može sabirati i varijable . Dok god su te varijable brojčane (mogu biti i decimalne, nije važno), sabiranje će biti logično. No probamo li pokrenuti ovaj program:

```
a = "Python"
b = "rules"
print a + b
```

rezultat je pomalo neočekivan, iako ne nelogičan: **Pythonrules**. Primetimo da su a i b u ovom slučaju stringovi (upišete li **a = python** (bez navodnika oko reči python), Python će misliti da varijablu "a" pokušavate pridružiti vrednosti varijable **python**, a budući da ta varijabla ne postoji, izbaci pogrešku). Python radi nešto što se zove združivanje stringova, iliti spajanje. Tu Python jednostavno na kraj vrednosti prve varijable ubacuje vrednost druge varijable. Python združivanje stringova radi automatski isključivo sa stringovima, a sa brojevima se to može uraditi tako da broj pretvorimo jednom naredbom (koju ćemo učiti neki drugi put) u string, i onda ih pridružimo.

Ukoliko želimo da ostavimo korisniku mogućnost da unese proizvoljne vrednosti za a i b, program ćemo pisati ovako:

```
a = input ("unesite vrednost a=")
b = input ("unesite vrednost b=")
c = a+b
print "a+b=",c
```

Ovaj programčić prvo pita korisnika da unese vrednost za a i b, a zatim promenjivoj c dodeljuje vrednost zbira i konačno štampa tekst "a+b=", iza kojeg postavlja vrednost promenjive c.

Još malo o print naredbi...

Ako ste pomno pratili tutorial, mogli ste videti nešto što je prilično samoobjašnjavajuće, ali što ipak do sada nismo objasnili. To je ova linija:

```
print "ja koristim", website, "da bi saznao stvari o muzici"
```

Prvo imamo print naredbu za koju znamo što radi. Nakon toga print naredba ispisuje string "ja koristim". Tu stane i traži vrednost varijable "website" i umeće je na to mjesto (uz razmake na početku i na kraju - tako da se oko toga ne trebate brinuti), i onda opet nastavlja ispis stringa "da bi saznao stvari o muzici". Ovo je vrlo korisno jer se u drugim programskim jezicima mora pribegavati drugačijim i mnogo komplikovanijim rješenjima.

Još malo o stringovima

Stringovi poseduju indekse koji određuju položaj pojedinih karaktera. Stringovi se za razliku od listi (koje ćemo kasnije objasniti) ne mogu menjati preko indeksa.

Pogledajmo ovaj primer:

```
string = "python"
print string[0:2]
```

Daje: **py** - prva dva slova stringa... Mnogo bolje nego QBasicov **LEFT$(a,b)** i slično. Takođe možemo uzimati vrednosti i s kraja (ovo važi i za stringove i za liste) sa predznakom -:

```
string = "python"
print string[-1]
```

Daje: **n**, zadnje slovo stringa.

Možda bi još koji primer pojasnio razliku između stringa, komentara i numeričkih vrednosti u Python-u pa evo ih:

PR.1:

```
#Ovo je prvi komentar
mleko=1     #Ovo je drugi komentar
             #a evo i treceg
string= " ovo nije komentar "
```

PR.2:

```
#ovo je komentar u programskoj liniji
print 2+2 #ovo je komentar u istoj programskoj liniji sa instrukcijom
```

PR.3:

```
#ovo je pocetak programa
print 2+2 #ova programska linija stampa 4
#print 2+2 ovo je samo komentar
```

PR.4:

```
# ovo je program koji deli dva broja
x = input("x=")     #sada pitamo korisnikakolika je vrednost za x
y=input("y=")       # sada pitamo korisnikakolika je vrednost za y
print "y/x=", y/x   #na kraju štampamo kolicnik
```

Probajte ovo:

```
>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp'
>>> word[:2] # The first two characters
'He'
```

```
>>> word[2:] # All but the first two characters
'lpA'
```

Najbolji način je da razmišljate o indeksima kao o tačkama između karaktera, ovako:

```
+---+---+---+---+---+
| H | e | l | p | A |
+---+---+---+---+---+
0 1 2 3 4 5
-5 -4 -3 -2 -1
```

Napravite novi string kormbinacijom sa starim:

```
>>> 'x' + word[1:]
'xelpA'
>>> 'Splat' + word[4]
'SplatA'
```

Možete se igrati, pa isprobajte npr nešto ovako:

```
>>> word[:2] + word[2:]
'HelpA'
>>> word[:3] + word[3:]
'HelpA'
>>> word[1:100]
'elpA'
>>> word[10:]
''
>>> word[2:1]
''
```

Indeksi mogu biti negativni ako počnemo brojati s desna:

```
>>> word[-1] # The last character
'A'
>>> word[-2] # The last-but-one character
'p'
>>> word[-2:] # The last two characters
'pA'
>>> word[:-2] # All but the last two characters
'Hel'
>>> word[-0] # (since -0 equals 0)
'H'
>>> word[-100:]
'HelpA'
>>> word[-10] # error
Traceback (most recent call last):
File "<stdin>", line 1, in ?
IndexError: string index out of range
```

A ovako možete proveriti koliko karaktera ima neka promenjiva:

```
>>> s = ’supercalifragilisticexpialidocious’
>>> len(s)
34
```

PRIMERI I VEŽBE

Primer 1 :

Napisati program koji će dati ovakav rezultat:

Rešenje:

```
ODE TO COMPUTERS
**********
```

```
I think I shall never see

A computer program smarter than me.

But computer's power is extremely fast,

It computes answer very fast!
```

Primer 2:

Napisati Python program koji pita korisnika kako se zove i koliko ima godina, a zatim te podatke štampa.
Rešenje:

```
ime=raw_input("kako se zoves ")
prez=raw_input("kako se prezivas ")
god=input("koliko imas godina ")
print "Ti si ",ime, prez," imas ", god," godina"
```

Upotreba Pythona kao kalkulatora

Pokušajmo da koristimo Python kao kalkulator, pa jednostavno, interpeter deluje isto kao i kalklator : ukucajte izraz i dobićete rezultat. Prioritet operatora je sličan kao u drugim programskim jezicima. Python izvršava operacije prema prioritetu koji možete kontrolisati postavljanjem zagrada. Kao u drugim programskim jezicima izrazom *imepromenjive=izraz* dodeljujete vrednost promenjivoj. Sve će biti jasnije sa primerima.

Evo prioriteta svih operatora:

+x, -x, ~x	Unary operators
x ** y	Power (right associative)
x * y, x / y,x % y	Multiplication, division, modulo
x + y, x - y	Addition, subtraction
x << y, x >> y	Bit shifting

x & y Bitwise and
x | y Bitwise or
x < y, x <= y, x > y, x >= y, x == y,
x != y, x <> y, x is y, x is not y, x
in s, x not in s<
Comparison, identity, sequence membership tests
not x Logical negation
x and y Logical and
lambda args: expr Anonymous function

Ukoliko u izrazu koristite više operatora istog prioriteta Python izvršava operacije sleva na desno, evo primera:

```
>>> 12/3*4
16
```

Zapažate da je rezultat 16, a ne 1. Primenite zagrade i evo 1.

```
>>> 12/(3*4)
1
```

Pogledajmo sledeće :

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

>>> # Celobrojno deljenje
>>> 7/3
2
```

ovako dajemo do znanja da želimo decimalni rezultat

```
>>> 7./3
2.3333333333333335
```

i još primera:

```
>>> 3*3.75/1.5
7.5
>>> 7.0/2
3.5
```

dodelite istu vrednost za više varijabli :

```
>>> x=z=y=0
>>> x
0
>>> z
0
>>> y
0
```

Pogledajmo razliku u sledećim primerima:

```
>>> 4**3./2
32.0
```

I drugi:

```
>>> 4**(3./2)
8.0
```

U prvom slučaju 4 prvo stepenujemo sa 3 , pa rezultat delimo sa 2, dok u drugom osnovu 4 stepenujemo na 3/2. Ukoliko želimo da korenujemo neki broj to možemo uraditi jednostavnim stepenovanjem na 1/2 ili ovako:

```
from math import*
x=input ("unesi x= ")
print "x=",x,"koren x=",sqrt(x)
```

u prvoj programskoj liniji pozivamo matematičke funkcije, u drugoj tažimo broj od korisnika, u trećoj ispisujemo rezultat dobijen primenom fje sqrt( ) onosno kvadratnog korena. Koje još f-je vam stoje na raspolaganju možete pogledati ukoliko u *Python Shell* ukucate help( ),math. Evo šta dobijete:

```
help> math
Help on built-in module math:

NAME
    math

FILE
    (built-in)

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    atan(...)
        atan(x)
```

Return the arc tangent (measured in radians) of x.

atan2(...)
atan2(y, x)

Return the arc tangent (measured in radians) of y/x.
Unlike atan(y/x), the signs of both x and y are considered.

ceil(...)
ceil(x)

Return the ceiling of x as a float.
This is the smallest integral value >= x.

cos(...)
cos(x)

Return the cosine of x (measured in radians).

cosh(...)
cosh(x)

Return the hyperbolic cosine of x.

degrees(...)
degrees(x) -> converts angle x from radians to degrees

exp(...)
exp(x)

Return e raised to the power of x.

fabs(...)
fabs(x)

Return the absolute value of the float x.

floor(...)
floor(x)

Return the floor of x as a float.
This is the largest integral value <= x.

fmod(...)
fmod(x,y)

Return fmod(x, y), according to platform C. x % y may differ.

```
frexp(...)
    frexp(x)

    Return the mantissa and exponent of x, as pair (m, e).
    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

hypot(...)
    hypot(x,y)

    Return the Euclidean distance, sqrt(x*x + y*y).

ldexp(...)
    ldexp(x, i) -> x * (2**i)

log(...)
    log(x[, base]) -> the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.

log10(...)
    log10(x) -> the base 10 logarithm of x.

modf(...)
    modf(x)

    Return the fractional and integer parts of x.  Both results carry the sign
    of x.  The integer part is returned as a real.

pow(...)
    pow(x,y)

    Return x**y (x to the power of y).

radians(...)
    radians(x) -> converts angle x from degrees to radians

sin(...)
    sin(x)

    Return the sine of x (measured in radians).

sinh(...)
    sinh(x)

    Return the hyperbolic sine of x.

sqrt(...)
    sqrt(x)
```

```
        Return the square root of x.

    tan(...)
        tan(x)

        Return the tangent of x (measured in radians).

    tanh(...)
        tanh(x)

        Return the hyperbolic tangent of x.

DATA
    e = 2.7182818284590451
    pi = 3.1415926535897931
```

PRIMERI I VEŽBE

Primer 1:

Napisatii program za Pitagorinu teoremu (izračunati hipotenuzu za unete vrednosti kateta).

Rešenje:

```
#Pitagora
a=input("unesite vrednost za katetu a=")
b=input("unesite vrednost za katetu b=")
c=(a**2+b**2)**(1.0/2)
print "Hipotenuza c=",c
```

Primer 2:

Napisati Python program koji računa površinu i obim kružnog isečka.

Rešenje:

```
#Kuzni isecak
r= input ("unesi poluprecnik r=")
alfa= input ("unesi ugao alfa=")
l=2*r+alfa*r
s=(r**2*alfa)/2
print "Povrsina kruznog isecka s=",s,", a obim l=",l
```

Primer 3:

Napisati Python program koji računa površinu i obim kružnog isečka pri čemu je potrebno da korisnik unese ugao u stepenima, a ne uradijanima kao u predhodnom primeru.

Rešenje:

```
#Kuzni isecak
r= input ("unesi poluprecnik r=")
alfastep= input ("unesi ugao u stepenima alfa=")
from math import pi
alfarad=(alfastep/180.0)*pi
l=2*r+alfarad*r
s=(r**2*alfarad)/2.0
print "Povrsina kruznog isecka s=",s,", a obim l=",l
```

2. Uslovno grananje, petlje i poravnanje

U prošlom nastavku Python tutoriala za smrtnike naučili smo par osnovnih komandi od kojih su najvažnije **input** i **print**. To su verovatno i naredbe koje ćete i najčešće koristiti. Sada možemo da pišemo programe koji će obavljati jednostavne akcije (računanje i isipis) i biti u stanju da prime ulaznu vrednost od korisnika. Ovo je korisno, ali smo i dalje ograničeni na na takozvano sekvencijalno izvršavanje komandi, to jest – moraju biti izvršene u zadatom redosledu…

Danas ćemo naučiti još par korisnih i često korištenih naredbi, a to su naredbe za uslovno grananje i programske petlje, koje su korisne kada želimo ponavljati iste naredbe više puta.

if naredba

Uslovno grananje nije ništa drugo nego programerski izraz za izvršavanje nekog koda na osnovu nekog uslova.

Da bi stvari bile jasnije, pogledajmo jedan primer:

```
ime = "Nikola"
if (ime == "Nikola"):
    print "Dobrodosao, Nikola"
```

U prvoj liniji definišemo varijablu **ime** i postavljamo joj vrednost **Nikola**. Vrednost smo okružili navodnicima jer je to string. U drugoj liniji zapravo proveravamo da li je vrednost varijable **ime** jednaka **Nikola**. Ako jeste, izvršava se kod koji je ispod naredbe **if** i koji obavezno mora biti odmaknut jednim TAB-om (o tome više čitajte pri kraju ovog članka). Tako je sintaksa naredbe if ovakva:

```
if (uslov):
    kod koji treba izvršiti ukoliko je uslov zadovoljen
```

Umesto proveravanja jednakosti, možemo proveravati i za ostale uslove. Evo popisa tih

uslova (tj. logičkih operatora):

Uslov što proverava

== jednako
!= nejednako
> veće
< manje
>= veće ili jednako
<= manje ili jednako

Pogledajmo još par primera:

```
a = 2
b = 3
if (a > b):
    print "a je vece od b"


a = 5
b = 5
if (a <= b):
    print "a je manje ili jednako b"
```

Ukoliko ste pomno pratili ovaj tutorial, primijetićete da je neki kod odmaknut od ruba stranice (tj. poravnat desno od koda iznad njega), npr. ovako:

```
if (a == 7):
    print "aha" # na ovo poravnanje mislimo
```

Neki programski jezici za definisanje bloka koda koriste { i } (vitičaste zagrade). Npr., u C++-u bi naredbu if-else pisali ovako:

```
if(name == Nikola)
{
   cout<<"Bravo, vase je ime Nikola"<<endl;
}
else
{
   cout<<"Vase ime nije Nikola"<<endl;
}
```

U C++-u poravnanje nije potrebno (zbog vitičastih zagrada). Python umesto vitičastih zagrada koristi poravnanje. Zamislite da je sintaksa Pythona ovakva:

```
if (uslov):
print "haha"
```

Kako bi interpreter znao kada blok naredbi koje se trebaju izvršiti ukoliko je uslov zadovoljen prestaje? Vi ne morate koristiti else, a čak i kada bi ga koristili, kada bi interpreter znao kada blok else prestaje? Zbog toga se koriste poravnanja, i ona se **moraju** koristiti (deo su Pythonove sintakse).

Napomena : Uvlačenje pasusa je važno u Python-u. Blokovi u uslovnom izvršavanju (i petlje i definicije funkcija – vidi ispod) moraju biti uvučeni (i uvlačenje mora biti jednako;<tab> se broji kao 8 mesta) tako da interpreter zna gde počinju i gde se završavaju. To takođe čini program lakšim za čitanje od strane ljudi.

if-else

If je vrlo korisna naredba, no ako postavljeni uslov nije zadovoljen, na ekranu se ne pokaže ništa. Tu dolazi u igru naredba **else** koja govori što će se dogoditi ako se uslov postavljen u **if** naredbi ne zadovolji.

Sintaksa ide ovako:

```
if (uslov):
    kod koji se izvršava ako je uslov zadovoljen
else:
    kod koji se izvršava ako uslov nije zadovoljen
```

Pogledajte ovaj primer:

```
ime = "Nikola"
if (ime == "Nikola"):
    print "Zdravo, Nikola!"
else:
    print "Zdravo, gost!"
```

Naravno, ovaj kod smo mogli napisati i tako da ime dobijemo od korisnika:

```
ime = raw_input("Unesite vase ime: ")
if (ime == "Nikola"):
    print "Zdravo, Nikola!"
else:
    print "Zdravo, gost!"
```

Ovaj program proverava da li je vrednost varijable **ime** jednaka **Nikola**. Ako jeste, ispisuje se **Zdravo, Nikola!**, u protivnom se ispisuje **Zdravo, gost!**.

Probajmo sa kuvarskim problemom, ako želimo da kažemo kompjuteru da proveri stanje salate na ringli? Ako je zagrejana, onda je treba skinuti – u suprotnom, treba je kuvati još minut ili dva. Kako to izražavamo?

Ono što želimo, je da kontrolisemo tok programa. Može da ide u dva pravca – ili da skloni šerpu, ili da je ostavi još malo u zavisnosti dali je dovoljno zagrejano. Ovo se zove uslovno izvršenje. To možemo ovako da obavimo:

```
temperatura = input("Koja je temperatura jela?")
if temperatura > 50:
....print "Salata je skuvana."
else:
....print "Kuvaj jos malo."
```

Značenje ovoga je očigledno : Ako je temperatura viša od 50 (stepeni), onda ispiši poruku korisniku da je jelo skuvano, u suprotnom , ispiši da kuva još malo.

Da se vratimo našem izračunavanju površine. Vidite li šta ovaj program radi?

```
# Program za racunanje povrsine

print "Dobro dosli u program za racunanje povrsine"
print "---------------------------------------"
print

# Ispisi izbor:
print "Molim vas izaberite oblik:"
print "1 Pravougaonik"
print "2 Krug"

# Unos korisnikovog izbora:
oblik = input("> ")

# Izracunavanje povrsine:
if oblik== 1:
....visina = input("Molim vas unesite visinu: ")
....sirina = input("Molim vas unesite sirinu: ")
....povrsina = visina*sirina
....print "Povrsina je", povrsina
else:
....poluprecnik = input("Molim vas unesite poluprecnik: ")
....povrsina = 3.14*(poluprecnik**2)
....print "Povrsina je", povrsina
```

Nove stvari u ovom primeru: print korišćeno samostalno ispisuje prazan red == proverava da li su dve stvari jednake, suprotno od =, što dodeljuje vrednost desne strane promenljivoj na levoj strani. ovo je važna lekcija ! ** je u Python-u moćan operator – tako se poluprečnik na kvadrat piše **2. Print može da ispiše više od jedne stvari. Samo ih odvojite zarezima. (Na ekranu prilikom ispisa će biti odvojeni jednim praznim mestom.) program je veoma jednostavan : Traži da se unese broj, koji govori da li korisnik želi da računa površinu kruga ili pravougaonika. Onda , koristi ako-izjavu (uslovno izvršenje , if=

ako) da odluči koji blok će biti korišten za računanje površine. Ta dva bloka su po građi istovetna sa onima korišćenim u prethodnim primerima. Uočite kako komentari čine kod lakšim za čitanje. Rečeno je da je prva zapovest programiranja : "Pravi komentare!" Bilo kako, to je lepa navika koju treba usvojiti.

Uslovno grananje s više uslova: if-elif-else

If-else je OK stvar ukoliko imamo samo jedan uslov koji treba proveriti. Ukoliko treba proveriti više uslova, možemo koristiti ili hrpu if naredbi, ili vrlo korisnu naredbu **elif**. Elif je skraćenica od "else if". Sa elif proveravamo neki drugi uslov koji nije naveden u if naredbi.

Sintaksa ide ovako:

```
if (uslov):
    kod
elif (uslov):
    kod
elif (uslov):
    kod
...
else:
    kod
```

Vežba:

Proširi program odozgo tako da ukljucuje računanje površine kvadrata, gde korisnik unosi dužinu samo jedne stranice. Da bi ovo uradili, potrebno je da znate samo jednu stvar : Ako imate više od dva izbora, možete da napišete nešto kao:

```
if nesto== 1:
....# Uradi jednu stvar...
elif nesto== 2:
....# Uradi nesto drugo...
elif nesto== 3:
....# Uradi nesto potpuno drugacije...
else:
....# Ako sve drugo propadne...
```

ovde elif predstavlja tajanstveni kod koji znači "else if" :). Znači ako je nešto jedno, uradi jednu stvar; u suprotnom, ako je nešto drugo, uradi nešto drugo i tako. Možete poželeti da dodate druge mogućnosti u program – kao trouglove i mnogouglove. To je na vama

A evo i primera - jedan VRLO jednostavni kalkulator:

```
a = input("unesite prvi broj: ")
op = raw_input("unesite operaciju (+, -, *, /, pot): ")
b = input ("unesite drugi broj: ")

if (op == "+"):
    print a + b
elif (op == "-"):
    print a - b
elif (op == "*"):
    print a * b
elif (op == "/"):
    print a / b
elif (op == "pot"):
    print a ** b # ** je stepenovanje
else:
    print "Unesite pravilnu operaciju!"
```

U ovom slučaju kod ispod else se izvršava ukoliko niti jedan uslov nije zadovoljen. elif naredbi možemo imati koliko želimo.

PRIMERI I VEŽBE

Primer 1:

Napisati Python program koji računa površinu i obim kružnog isečka pri čemu je potrebno da korisnik unese ugao u stepenima do vrednosti 360 u suprotnom da obavesti korisnika da je unos pogrešan.

Rešenje:

```
#Kruzni isecak uslovno
r =input ("unesi poluprecnik r=")
alfastep = input ("unesi ugao u stepenima alfa=")
if alfastep<360:
    from math import pi
    alfarad=(alfastep/180.0)*pi
    l=2*r+alfarad*r
    s=(r**2*alfarad)/2.0
    print "Povrsina kruznog isecka s=",s,", a obim l=",l
else:
    print" Uneli ste pogresnu vrednost ugao alfa mora biti manji od 360 stepeni"
```

Primer 2:

Napisati Python program za sledeću funkciju

$$Y = \begin{cases} 0, X \le 0 \\ X^2, 0 < X \le 1 \\ 1, X > 1 \end{cases}$$

Rešenje:

```
# Y funkcija
x = input("Unesi x=")
if x<=0:
    y=0
elif x>1:
    y=1
else:
    y=x**2
print "y=",y
```

Programska petlja – while

Šta ako želimo ponoviti neki kod više puta? Jedan trivijalni primer za ovo bilo bio ispisivanje stringa "Zdravo, Nikola" sedam puta. Tu uskače naredba **while**. Za razliku od **if**, koji se izvršava kada uslov postane tačan, **while** se izvršava **dok god** je zadati uslov tačan.

Sintaksa:

```
while (uslov):
    kod koji se izvršava dok je uslov tačan
```

Pogledajmo primer:

```
a = 1
while (a <= 7):
    print "Zdravo, Nikola!"
    a = a + 1
```

Ovo bi ispisalo "Zdravo, Nikola!" sedam puta na ekranu. Analizirajmo malo ovaj kod:
- prva linija definiše varijablu, i dodeljuje joj vrednost 7,
- druga poziva naredbu **while** i definiše da se navedene naredbe izvršavaju dok god je a manje ili jednako 7,
- treća linija ispisuje string,
- četvrta uvećava varijablu **a** za 1 jer bi inače varijabla a bila uvijek manja ili jednaka 7, što bi dovelo do beskonačnog loopa.

Sledeći **While** loop je loop koji se ponavlja sve dok određeni uslov nije zadovoljen; recimo - sve dok **a** nije jednak 10 ili sve dok je **a = 10**.

Pogledajmo primer jednog jednostavnog while loopa:

```
a = 0
while a <= 10:
    print "Hello"
    a = a + 1
```

Iz ovoga dobijamo:

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Analizirajmo kod:

u prvoj liniji se varijabla a postavlja na vrednost nula. Drugu liniju je lako objasniti ako se pročita na običnom engleskom: **"While a less or equal 10"**. Znači, izvršavaj sledeće naredbe sve dok je a manji ili jednak 10.
Sledeća linija ispisuje string **"Hello"**, a sledeća uvećava varijablu za 1. Tu liniju smo mogli pisati i kao **a += 1**.

Pogledajmo što bi se dogodilo kad bi ispustili posljednju liniju - **a = a + 1**:

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
# još nekoliko desetaka puta ovako ;-)
# prekinuto sa CTRL+C
Traceback (most recent call last):
  File "<pyshell#3>", line 2, in ?
    print "Hello"
  File "C:\PYTHON22\Tools\idle\PyShell.py", line 679, in write
    self.shell.write(s, self.tags)
  File "C:\PYTHON22\Tools\idle\PyShell.py", line 670, in write
    raise KeyboardInterrupt
KeyboardInterrupt
```

Ovo se stručno naziva "infinite loop" - loop iz kojeg nema izlaska. Ako bolje pogledate kod videćete , budući da se varijabla a ne uvećava (konstantno ostaje 0), da iz loopa nema izlaska; tj. uslov se nikad ne zadovoljava.To rezultuje beskonačnim ponavljanjem zadatih naredbi.

Šta bi sa našim kuvarskim problemom? Ako želimo da proverimo jelo što puta, to je veoma dobro resenje; ali mi ne znamo je li to dovoljno – ili je previse. Mi samo želimo da nastavimo da proveravamo dok ne bude dovoljno (ili dok nije dovoljno vruće - zavisi od gledista). Zato koristimo while:

```
# Program za kuvanje

# Namesti pauziranje funkcije
from time import sleep

print "Molim vas pocnite da kuvate. (Vraticu se za 3 minuta.)"

# cekaj 3 minuta (to jest, 3*60 sekundi)...
sleep(180)

print "vratio sam se :)"

# koliko vruce je taman koliko treba?
dovoljno_vruce = 50
```

```
temperatura = input("koliko vruce je jelo? ")
while temperatura< dovoljno_vruce:
....print "Nije dovoljno... kuvaj to jos malo..."
....sleep(30)
....temperatura = input("OK. Koliko je sada? ")

print "Dovoljno je! zavrsili ste !"
```

Nove stvari u ovom primeru...

Neke korisne funkcije se čuvaju u modulima i mogu se uvesti. U ovom slučaju uvozimo (import) funkciju sleep (koja pauzira program na odredjen broj sekundi)iz modula time koji dolazi uz Python. (Moguće je i napraviti svoje module...)

Vežba:

Napišite program koji stalno unosi brojeve od korisnika i sabira ih dok ne dodje do sume od 100. Napišite drugi program koji unosi 100 brojeva od korisnika i ispisuje njihov zbir .

Postoji još jedna programska petlja, a to je **for**. No, da bi objasnili petlju **for**, treba prvo objasniti liste.

3. Liste, manipulacija listama i for loop

U prošlom nastavku Python tutoriala naučili smo kondicionalno grananje i while programsku petlju. Danas ćemo naučiti liste i FOR programsku naredbu. Da bi naučili FOR programsku naredbu moramo znati liste, pa krenimo logičnim redosledom... Do sada smo upoznali varijable - jedan tip podataka. One mogu sadržavati isključivo jednu vrednost:

```
ime = "Nikola"
```

Sada zamislite varijablu koja može da sadrži više vrednosti. Taj se tip podataka zove lista. Lista se uvijek okružuje uglatim zagradama: [i]. Pogledajmo primer jedne liste:

```
python = ["jednostavan", "objektno orjentisan", "lagan", "mocan"]
```

Ukoliko napišemo **print python** dobijamo ovo:

```
['jednostavan', 'objektno orjentisan', 'lagan', 'mocan']
```

Ukoliko nam treba samo prvi unos iz liste (u ovom slučaju "jednostavan"), koristimo zero-based indexing (iliti indeksiranje na bazi nule). Tako prvi unos ima indeks 0, drugi 1, treći 2, itd., tako da ako želimo izdvojiti samo prvi unos iz liste, pišemo ovo:

```
print python[0]
```

Broj u uglatim zagradama je indeks.

Python može sadržavati sve tipove varijabli u listi - i brojeve i stringove, pa čak i druge liste:

```
miks = ["Nikola", 1234, ["jos", "jedna", "lista"], 3.1415]
```

Manipulisanje listama

Liste se mogu sabirati sa + operatorom:

```
p2pprogs = ["kazaa lite", "kazaa", "winmx", "audiogalaxy"]
mp3sites = ["mp3.com", "listen4ever.com"]
music = p2pprogs + mp3sites
print music
```

dobijamo ovo:
['kazaa lite', 'kazaa', 'winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com']

Takođe možete uzimati i delove liste, npr. samo prva dva unosa u listu:

```
print music[0:2] # ispisuje unose sa indeksom 0 i 1
```

daje ovo:
['kazaa lite', 'kazaa']

Podaci u listama se mogu menjati:

```
music[1] = "kazaa media desktop" # ovo je zapravo drugi unos u listi
```

Sada naša lista izgleda ovako:
['kazaa lite', 'kazaa media desktop', 'winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com']

Ukoliko treba proveriti da li je neki unos u listi, dobro će vam doći naredbe "in" i "not in" koje vraćaju booleanski rezultat (true ili false - 0 ili 1):

```
print "kazaa lite" in music
```

daje: **1** (true), jer kazaa lite postoji u listi "music".

```
print "zweistein.web" not in music
```

daje: **1** (true) jer zweistein.web ne postoji u listi "music".

Ukoliko želite dodati unos u listu, naredba append() će vam pomoći:

```
music.append("zweistein.web")
```

A lista izgleda ovako:

['kazaa lite', 'kazaa media desktop', 'winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web']

Za brisanje se koristi naredba del() koja briše unos na osnovu indeksa:
del music[0]

Ovako izgleda lista nakon što smo obrisali unos sa indeksom 0:
['kazaa media desktop', 'winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web']

Takođe možemo brisati po vrednosti umesto po indeksu:
music.remove("kazaa media desktop")

Ovako izgleda naša lista nakon što smo obrisali unos sa vrednošću "kazaa media desktop":
['winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web']

Ukoliko želite poređati listu po abecedi, koristiće vam naredbe sort() i reverse():
music.sort()
# rezultat: ['audiogalaxy', 'listen4ever.com', 'mp3.com', 'winmx', 'zweistein.web']

music.reverse()
# rezultat: ['zweistein.web', 'winmx', 'mp3.com', 'listen4ever.com', 'audiogalaxy']

Još primera: (ukoliko nešto nije jasno pitajte me na času)

```
>>> a=[1,6.9,8]
>>> a
[1, 6.9000000000000004, 8]
>>> b=a
>>> b
[1, 6.9000000000000004, 8]
>>> b[0]=9
>>> a
[9, 6.9000000000000004, 8]
>>> b
[9, 6.9000000000000004, 8]
>>> g=a+b
>>> g
[9, 6.9000000000000004, 8, 9, 6.9000000000000004, 8]
>>> g[0]=80
>>> g
[80, 6.9000000000000004, 8, 9, 6.9000000000000004, 8]
```

```
>>> c=a*3
>>> c
[9, 6.9000000000000004, 8, 9, 6.9000000000000004, 8, 9, 6.9000000000000004, 8]
>>> x=[[1,3],[6,9]]
>>> x
[[1, 3], [6, 9]]
>>> x[0]
[1, 3]
>>> x[1]
[6, 9]
>>> x[0][0]
1
>>> x[0][1]
3
>>> x[0][0]=6
>>> x
[[6, 3], [6, 9]]
```

For loop

For loop nam služi da bi izveli neku naredbu na svakoj vrednosti u nekom tipu podataka (varijabli, listi...).

```
for hrana in "paprika", "jaja", "paradajz":
....print "Ja volim", hrana
```

ovo znači : za svaku stavku u listi "paprika", "jaja", "paradajz", ispiši da je voliš. Blok unutar petlje je izvršen po jednom za svaku stavku, i svaki put, trenutni element je dodeljen promenljivoj hrana (u ovom slucaju).

```
for broj in range(1,100):
....print "Zdravo, svete!"
....print "samo", 100 - broj, "je preostalo..."

print "zdravo, svete"
print "To je bio poslednji... Uh!"
```

Raspon funkcije (range) vraća listu brojeva u datom rasponu (uključujući prvi, bez poslednjeg... u ovom slučaju, [1..99]). Zato, da preformulišemo ovo:

Sadržaj petlje je izvršen za svaki broj u rasponu od (uključujući) 1 do (isključujući) 100. (Šta telo petlje u stvari radi se razlikuje od programa do programa i to je na vama)

Sintaksa je:

```
for privremena_varijabla in tip_podataka:
    blok naredbi
```

Pogledajmo jedan primer i sve će biti malo jasnije:

```
music = ['winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web']
for website in music:
    print "ja koristim", website, "da bi saznao stvari o muzici"
```

šta daje sljedeći ispis:

```
ja koristim winmx da bi saznao stvari o muzici
ja koristim audiogalaxy da bi saznao stvari o muzici
ja koristim mp3.com da bi saznao stvari o muzici
ja koristim listen4ever.com da bi saznao stvari o muzici
ja koristim zweistein.web da bi saznao stvari o muzici
```

For loop možete koristiti i za stringove:

```
string = "python tutorial"
for slovo in string:
    print slovo
```

Ovo daje sljedeći ispis:

```
p
y
t
h
o
n

t
u
t
o
r
i
a
l
```

Zanimljiv efekat, ne? :)
Spomenuli smo funkciju range() koja vam olakšava da manipulišete sa nizovima vrednosti. Ova f-ja generiše aritmetičku progresiju članova liste, da pojasnimo:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Zapažate da lista počinje 0? Moguće je da range započne drugim brojem ili da zadate korak (step) koji želite:

```
>>> range(5,10)
[5, 6, 7, 8, 9]
>>> range(0,10,5)
[0, 5]
>>> range(0,10,2)
[0, 2, 4, 6, 8]
>>> range(1,10,2)
[1, 3, 5, 7, 9]
>>> range(-10,-100,-30)
[-10, -40, -70]
```

Uz pomoć naredbe len( ) slika je još jasnija:

```
>>> a=['Mary','had','a','little','lamb']
>>> for i in range(len(a)):
        print i, a[i]


0 Mary
1 had
2 a
3 little
4 lamb
```

Tuple

Do sada ste upoznali dva tipa podataka - stringove i liste. String se ne može menjati, a lista može. Sada ćemo upoznati još jedan tip podataka, a to je tuple. Tuple je vrlo sličan listi - samo se podaci u tupleu ne mogu menjati.

Tuple definišemo ovako:

```
music = 'winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web'
```

Kao što vidimo, veoma slično listi, samo što nema uglastih zagrada. Zapravo, i tuplei koriste zagrade (ali ne uglaste, već male - (i)), ali one nisu potrebne. Tako gornji tuple možemo napisati ovako:

```
music = ('winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web')
```

Ukoliko želimo napraviti tuple sa samo jednim unosom (što je potrebno ako želimo već postojećem tupleu dodati nešto novo - objašnjeno kasnije), jednostavno dodamo zarez iza stringa:

```
music2 = ('uraniummusic.com',)
```

što daje isti rezultat kao gornji primer.

Za tuple važi gotovo sve kao i za liste:

```
>>> type(music) # sa funkcijom type možemo saznati kojeg nam je tipa neka varijabla - str, int, tuple, list...
<type 'tuple'>
>>> print music[0]
winmx
>>> print music[3]
listen4ever.com
>>> music = music + music2
>>> print music
('winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web', 'uraniummusic')
```

Ali, ukoliko pokušamo nešto menjati:

```
>>> del music[5]
Traceback (most recent call last):
  File "<input>", line 1, in ?
TypeError: object doesn't support item deletion
>>> music[5] = "lambgoat.com"
Traceback (most recent call last):
  File "<input>", line 1, in ?
TypeError: object doesn't support item assignment
```

Ili, ukoliko pokušamo pridružiti neki drugi tip podataka tupleu:

```
>>> music = ('winmx', 'audiogalaxy', 'mp3.com', 'listen4ever.com', 'zweistein.web')
>>> music2 = "uraniummusic"
>>> music = music + music2
Traceback (most recent call last):
  File "<input>", line 1, in ?
TypeError: can only concatenate tuple (not "str") to tuple
```

Tuplei su korisni za čuvanje podataka za koje ne želite da korisnici mijenjaju.

PRIMERI I VEŽBE

Primer 1 :

Napisati Python program za pronalaženje najmanljeg broja u skupu proizvoljno unetih brojeva.
Rešenje:

```
#Najmanji broj
n = input("koliko ima brojeva u skupu?")
xmin = input("Unesi prvi broj")
for i in range (2,n):
    x = input("unesi sledeci broj")
    if x<xmin:
       xmin=x
print "najmanji broj je ",xmin
```

Primer 2 :

Napisati Python program za funkciju $y=x^2$ u opsegu celobrojnih vrednosti x od početne a do krajnje b proizvoljnim celobrojnim korakom.

Rešenje:

```
 # y=x**2 od a do b korakom k
a=input("unesi pocetnu vrednost a=")
b=input("unesi pocetnu vrednost b=")
k=input("unesi  vrednost koraka k=")
for x in range(a,b+k,k):
    y=x**2
    print "y=",y
```

napomena: pitate se zašto u range ne stoji samo b, pa zato što Python u tom slučaju ne uzima poslednju vrednost b jer ona ne ulazi u opseg. Ukoliko bi umesto b+k stajalo samo b evo šta se događa (opet primer):

```
unesi pocetnu vrednost a=2
unesi pocetnu vrednost b=8
```

```
unesi  vrednost koraka k=2
y= 4
y= 16
y= 36
```

vidimo da u tom slučaju nije izračunato y za x=8, pa prema tome ako se poslužimo trikovima rezultat našeg koda bi trebao da izgleda ovako:

```
unesi pocetnu vrednost a=2
unesi pocetnu vrednost b=8
unesi  vrednost koraka k=2
y= 4
y= 16
y= 36
y= 64
```

Primer 3 :

Napisati Python program koji pita korisnika da unese broj i da ukoliko je broj pozitivan ispiše "pozitivan" i izračuna njegov koren, a ukoliko je negativan ispiše "negativan", promeni mu znak i izračuna koren.

Rešenje:

```
print"Progran za racunanje korena broja"
from math import*
x=input("unesite broj x=")
if x>0:
    print " uneti broj je pozitivan"
else:
    print "uneti broj je negativan"
    x=-x
koren=sqrt(x)
print "Kvadratni koren broja koji ste uneli je ",koren
```

Primer 4 :

Napisati program koji štampa sve parne a zatim sve neparne brojeve od 0 do 100.
Rešenje:

```
# parni i neparni brojevi
print "parni brojevi su:"
for i in range(0,101,2):
    print i
print"a ovo su neparni brojevi:"
for n in range(1,100,2):
    print n
```

Primer 5 :

Pokušajte napisati program sličan predhodnom ali ostavite korisniku izbor opsega brojeva.

Rešanje: ovo bi mogli sami pokušati

Primer 6 :

Napisati program koji računa srednju vrednost u skupa proizvoljnih brojeva koje korisnik unosi.

Rešanje:

```
# srednja vrednost
n =input("koliko brojeva ima u skupu?")
suma=0
for i in range(n):
    X =input("Unesite  broj x=")
    suma=suma+X
Xsr=suma/n
print " Srednja vrednost Xsr=",Xsr
```

Primer 7:

Napisati Python program koji pita korisnika da unese brojeve m i n proizvoljan broj puta i na kraju izračuna srednju vrednost proizvoda za sve unete parove vrednosti m i n.

Rešanje:

```
#srednja vrednost proizvoda
q =input("Koliko parova vrednosti zelite?")
s=0
for i in range (q):
    n =input("Unesite vrednost n=")
    m =input("Unesite vrednost m=")
    s=s+n*m
print "Suma svih proizvoda SUM(n*m)=",s
```

Primer 8 :
Napisati Python program za proračun pritiska gasa prema jednačini idealnog gasnog stanja, tako da korisnik unosi vrednosti za ostale parametre.

$$PV = nRT$$

Rešenje: ovo je zaista lako

Primer 9 :

Napisati Python program za proračun pritiska gasa prema jednačini idealnog gasnog stanja, tako da korisnik unosi vrednosti za ostale parametre, ali sada program treba izvrši onoliko proračuna koliko korisnik želi.

$PV = nRT$

Rešenje: ovo je slično, samo primenite for petlju...

Primer 10 :

Napisati Python program za proračun pritiska za niz ekvidistantnih zapremina od vp do vk celobrojnim korakom dv, prema VDW jednačini:

$$p = \frac{RT}{v-b} - \frac{a}{v^2}$$

Rešenje:

```
#racunanje pritiska prema VDW j-ni
R=8.314
a= input("Unesi parametar a=")
b= input("Unesi parametar b=")
T= input("Unesi temperaturu u K T=")
vp= input("Pocetna zapremina vp=")
vk= input("Krajnja zapremina vk=")
dv= input("Celobrojni korak dv=")
print"----------------------------"
for v in range(vp,vk,dv):
    p=R*T/(v-b)-a/v/v
    print "za v=", v ,"p=", p
```

Primer 11 :

Napisati Python program za proračun pritiska za više proizvoljnih vrednosti zapremina, prema VDW jednačini:

$$p = \frac{RT}{v-b} - \frac{a}{v^2}$$

Rešenje:

```
#racunanje pritiska prema VDW j-ni
R=8.314
n= input("Za koliko unetih zapremina zelite proracun?")
a= input("Unesi parametar a=")
b= input("Unesi parametar b=")
T= input("Unesi temperaturu u K T=")
print"------------------------"
for i in range(n):
    v=input("unesi zapreminu v=")
    p=R*T/(v-b)-a/v/v
    print "za v=",v ,"p=",p
```

Primer 12:

Napisati **Python** program za proračun viskoziteta date supstance μ, i proizvoljan broj, *n* temperatura *T* . Pri tome se svi potrebni podaci unose sa tastature. Na ekran prikazati temperature *T*, odgovarajuće viskozitete μ i standardno kvadratno odstupanje izračunatih viskoziteta s_μ^2.

Polazni podaci su: ime supstance , molekulska masa *M*, kritičan pritisak P_C, kritična temperatura T_C. Da bi izračunali viskozitet, potrebno je izračunati redukovanu temperaturu T_r, koja se računa po formuli:

$$T_r = \frac{T}{T_c}$$

pri čemu mora biti zadovoljen uslov da je $T_r \leq 1.5$. Tada viskozitet μ računamo po formuli:

$$\mu = 34.0 \cdot 10^{-5} \cdot \sqrt{M} \cdot T_r^{0.94} \cdot \frac{\sqrt[3]{P_c^2}}{\sqrt[6]{T_c}}$$

Standardno kvadratno odstupanje viskoziteta se računa kao:

$$s_\mu^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n}\mu_i^2 - \frac{\left(\sum_{i=1}^{n}\mu_i\right)^2}{n}\right]$$

gde su μ_i, viskoziteti supstance na temperaturi T_i, (i = 1,...,n).

Program testirati za n-pentan: M=71.15, P_C=33.3, T_C=469.5 i sledeće temperature: T=198, 298, 398, 498, 598, 698

Rešenje:

```
#Viskozitet
print" Ovo je proracun viskoziteta supstance za proizvoljan broj temperatuura"
ime=raw_input("Unesi  ime supstance ")
M=input("Unesi M=")
Pc=input("Unesi Pc=")
Tc=input("Unesi Tc=")
n=input("Za koliko temperatura zelis proracun? ")
mi=n*[0]
T=n*[0]
rt=n*[0]
s1=0
s2=0
for i in range(n):
    while True:
        T[i]=input("Unesi T=")
        rt[i]=T[i]/Tc
        if rt[i]<=1.5:
            mi[i]=34.3*10**-8*rt[i]**0.94*M**1/2*Pc**2/3/Tc**1/6
            s1=s1+mi[i]
            s2=s2+mi[i]**2
            break
        else :
            print"uneli ste pogresnu vrednost, probajte ponovo"

sm2=(s2-(s1**2/n))/(n-1)
print" Za supstancu ", ime
for i in range (n):
    print "Za temperaturu T= ",T[i]," izracunati viskozitet mi=",mi[i]
print"Standardno kvadratno odstupanje viskoziteta Sm2=",sm2
```

***NAPOMENA:*Break naredba omogućuje izlaz iz for i while petlji, dok Continue naredba nastavlja ponavljanje petlji.**

Nove stvari u ovom primeru: Zapazili ste da smo za veličine koje treba računati proizvoljan broj puta pisali mi=n*[0], T=n*[0], rt=n*[0], odnosno svakoj promenjivoj smo dodeljivali n puta listu [0].

Primer 20 :

Napisati **Python** program za proračun pada pritiska Δp pri protoku tečnosti kroz cev za proizvoljan broj n protoka Q. Na ekran prikazati protoke i izračunate padove pritisaka kao i standardnu devijaciju protoka s_Q.

<u>*Polazni podaci su*</u>: ρ (gustina), μ (viskozitet), d (prečnik cevi), L (dužina cevovoda)

Da bi izračunali pad pritiska, potrebno je izračunati sledeće:

- poprečni presek: $S = \frac{d^2\pi}{4}$
- srednju brzinu: $v = \frac{Q}{\rho \cdot S}$
- Rejnolds-ov broj: $R_e = \frac{v \cdot d \cdot \rho}{\mu}$ i

- koeficijent podužnog trenja λ koji je funkcija R_e i definisan je na sledeći način:

$$\lambda = \begin{cases} 2100 \le R < 3500 & 0.0001 \cdot R_e^{0.575} \\ R_e < 2100 & \frac{64}{R_e} \\ R_e \ge 3500 & 0.035 + 2.264 \cdot R_e^{-0.428} \end{cases}$$

Pad pritiska Δp se računa po formuli:

$$\Delta p = \lambda \cdot \frac{L}{d} \cdot \frac{\rho \cdot v^2}{2} \quad \text{[kPa]}$$

Standardna devijacija protoka s_Q se računa kao:

$$s_Q = \sqrt{\frac{n \cdot \sum_{i=1}^{n} Q_i^2 - \left(\sum_{i=1}^{n} Q_i \right)^2}{n \cdot (n-1)}}$$

Gde su Q_i (i = 1,...,n) uneti protoci tečnosti.
Tetirati program za sledeće podatke: : ρ = 1000, μ = 1.8×10^{-5}, d = 0.1, L = 100 i sledeće protoke
Q = 0.01, 0.1, 1, 5, 10

Rešenje:

```
#PAD PRITISKA U CEVI
print"Proracun pada pritiska pri protoku tecnosti kroz cev"
ro =input("Unesi gustinu ro=")
mi =input("Unesi viskozitet mi=")
d =input("Unesi precnik cevi d=")
l =input("Unesi duzinu ceviovoda l=")
```

```
n =input("Koliko proracuna zelite?")
s =d**2*3.14/4
q =n*[0]
v =n*[0]
re =n*[0]
la =n*[0]
dp =n*[0]
s1 =0
s2 =0
for i in range(n):
    q[i] =input("unesi protok Q=")
    v[i] =q[i] / (ro*s)
    re[i] =v[i]*d*ro / mi
    if re[i] < 2100:
        la[i] = 64 / re[i]
    elif re[i] >= 3500:
        la[i] =0.035+2.264*re[i]**-0.428
    else:
        la[i]= 0.0001*re[i]**0.575
    dp[i]= la[i]*l/d*ro*v[i]**2/2
    s1=s1+q[i]
    s2=s2+q[i]**2
sdq = (n*s2-s1**2 / (n*(n-1)))**1/2
for i in range (n):
    print "Za unetu vrednost protoka Q=",q[i]," izracunati pad pritiska je dP=",dp[i]
print "Standardna devijacija protoka je Sq=",sdq
```

5. Veći programi – Skraćivanje posla,funkcije

Ako želite kratak uvid u knjigu, nećete čitati sve stranice – pogledaćete sadrzaj, zar ne? To je jednostavno spisak glavnih tema u knjizi. Sada - zamislite pisanje kuvara. Mnogi recepti, kao "makaroni u kremastom prelivu " i "Svajcarska pita sa prelivom" mogu sadrzati slične stvari, kao preliv, u ovom slučaju - ipak vi ne bi želeli da u svakom receptu ponovo pišete kako se pravi preliv. (OK...vi zaista ne pravite preliv... ali držimo se njega u našim primerima :)). Stavićete recept za preliv u odvojeno poglavlje, i jednostavno ukazati na njega iz drugih recepata. Tako – umesto pisanja čitavog recepta svakog puta, treba da koristite samo ime poglavlja. U kompjuterskom programiranju to se zove poziv procedure.

Jesmo li naleteli na nešto slično ovome ranije? Da. Umesto da smo kompjuteru tačno objasnili kako da dobije odgovor od korisnika(OK - ne možemo baš to da uradimo... ali ne možemo ni da napravimo preliv.. :)) jednostavno smo koristili input - predodređenu funkciju. Zapravo, možemo sami da napravimo svoje funkcije da ih koristimo kao procedure.

Recimo da želimo da nadjemo najveći intidžer (pozitivan broj bez ostatka) koji je manji od zadatog broja. Na primer, ako zadamo broj 2,7 taj inidzer će biti 2. Ovo se često zove "pod" (floor) datog broja. (Ovo se u stvari može obaviti preko ugradjenih Python-ovih funkcija, ali opet...) kako ćemo uraditi ovo? Jednostavno rešenje je da probamo sve mogućnosti počev od nule:

```
broj = input("Koji je broj u pitanju? ")

floor = 0
while floor < broj:
....floor = floor+1
floor = floor-1

print "Pod broja", broj, "je", floor
```

Primetićete da se petlja završava onda kada pod više nije manji od broja; dodali smo jedan previše. Zbog toga moramo da oduzmemo jedan. Šta ako želimo da koristimo ovu "floor"-stvar u složenom matematičkom izrazu? Morali bi da pišemo petlju za svaki broj kome treba naći "pod". Ne baš lepa situacija... Pogadjate šta ćemo uraditi umesto toga : Stavićemo sve u našu novu funkciju nazvanu "floor" :

```
def floor(broj):
....rezultat = 0
....while rezultat < broj:
........rezultat = rezultat+1
....rezultat = rezultat-1
....return rezultat
```

Nove stvari u ovom primeru....

1. Funkcije se definisu ključnom reči def, koju prati njeno ime i očekivani parametri u zagradi.
2. Ako funkcija treba da vrati neku vrednost, ovo se postiže ključnom reči return (koja istovremeno automatski prekida funkciju.

Sad kad smo je definisali možemo je koristiti, ovako:

```
x = 2.7
y = floor(2.7)
```

Posle ovoga, Y bi trebalo da ima vrednost 2. Takođe je moguće napraviti funkcije sa više od jednog parametra :

```
def sum(x,y):
....return x+y
```

Napišite funkciju koja sadrzi Euklidov metod za nalazenje zajedničkog množioca dva broja. To ide ovako:

1. Imate 2 broja, a i b, gde je a veće od b
2. Ponavljate sledeće dok b ne postane nula:
    1. a se menja u vrednost b
    2. b je promenjeno u podsetnik kada je a (pre promene) podeljeno sa b (pre promene)
3. Na kraju vratire sa return poslednju vrednost broja a

Ovde je data sažeta verzija algoritma :

```
def euklid(a,b):
....while b:
.......a,b = b,a % b
....return a
```

Saveti:

- Koristi a i b kao parametre funkcije
- Jednostavno pretpostavi da je a veće od b
- Podsetnik kad je x podeljeno sa z se računa kao izraz x % z
- Dve promenljive mogu biti istovremeno dodeljene, kao : x, y = y, y+1. Ovde je x primilo vrednost y (to jest, vrednost koju je Y imalo pre dodele) a ipsilon je uvećano za jedan.

Vrsta skraćivanja posla koju smo koristili pravljenjem funkcija zove se proceduralno skraćivanje, i mnogi jezici koriste reč procedura pored reči funkcija. U stvari, ova dva koncepta su različita, ali se oba zovu funkcije u Python-u (pošto se manje-vise definisu i koriste na isti način)

Koja je razlika (u drugim jezicima) izmedju funkcija i procedura? Pa, kao što ste primetili u prethodnom odeljku, Funkcije mogu da vrate neku vrednost. Razlika lezi u tome što procedure ne vraćaju nikakvu vrednost. Na mnogo načina ova podela funkcija u dve grupe –na one koje vraćaju i one koje ne vraćaju vrednost – može biti veoma korisna.

Funkcija koja ne vraća vrednost ("procedura") se koristi kao pod-program ili podrutina. Pozovemo funkciju i program uradi neku stvar, na primer umuti slag ili slično. Ovakve

funkcije možemo koristiti na mnogo mesta bez ponovnog pisanja koda. (Ovo se zove ponovno korisćenje koda, vratićemo se na to kasnije.)

Korisnost funkcija (ili procedura) leži u njihovim sporednim efektima – menjaju svoju okolinu (mešajući sećer i kremu i muteći ih, na primer...) Pogledajmo ilustraciju :

```
def zdravo(kome):
....print "Zdravo,", kome

zdravo("svete")
# Ispisuje na ekran "Zdravo, svete"
```

Ispisivanje stvari smatra se sporednim efektom, a pošto je to sve što ova funkcija radi, to je tipično za takozvane procedure. Ali... Ona ne menja stvarno svoju okolinu, zar ne? Kako bi mogli to da uradimo? Da probamo:

```
# *Pogresan nacin* da se to uradi
starost= 0

def setStarost(a):
....starost= a

setStarost(100)
print Starost
# Ispisuje "0"
```

Šta je pogrešno ovde? Problem je što funkcija setStarost stvara svoju lokalnu promenljivu,takođe nazvanu Starost koja je vidljiva samo unutar setStarost. Kako da to izbegnemo? Možemo da koristimo nešto što se zove globalna promenljiva.

Napomena : Globalne promenljive se ne koriste mnogo u Python-u. One lako vode do loše strukture, ili onoga što zovemo špageti-kod. Ovde ih koristimo da vas upoznamo sa komplikovanijim tehnikama - molim vas da ih izbegavate ako možete.

Govoreći interpreteru da je promenljiva globalna (radi se izjavom kao Global Starost) mi mu govorimo da koristi promenljivu izvan funkcije umesto da stvori novu. (Pa je to u stvari globalna kao suprotno od lokalna). Program možemo ispraviti ovako:

```
# Ispravan, ali ne bas najbolji način
Starost= 0

def setStarost(a):
....global Starost
....Starost= a

setStarost(100)
print Starost
# Ispisuje "100" na ekran racunara.
```

Kada naučite o objektima (ispod), videćete da bi priličniji način da se ovo uradi bio da se koristi objekt sa Starost osobinom i setStarost metodom. U odeljku o strukturi podataka, takođe će te videti bolje primere za funkcije koje menjaju svoju okolinu.

Pa - šta je sa realnim funkcijama? šta je funkcija uopste? Matematičke funkcije su kao mašine koje imaju ulaz i računaju rešenje. Vratiće isti rezultat svaki put, kada je ista unesena vrednost. Na primer:

```
def kvadrat(x):
....return x*x
```

ovo je isto kao matematička funkčija f(x)=x na kvadrat. Ponasa se fino, radi samo na ulaznoj promenljivoj i ne menja svoju okolinu u bilo kom pogledu.

I tako – zaokružili smo dva načina za pravljenje funkcija : Jedan tip je više kao procedure, ne vraća rezultat; drugi je više kao matematička funkcija i ne radi ništa osim što vraća rezultat (skoro ništa). Naravno, moguće je napraviti nešto izmedju, iako čim funkcija menja nešto, jasno je da to radi. Možete ovo objaviti njenim imenom, na primer korisćenjem imenica za čiste funkcije kao kvadrat i imperativni oblik za procedurolike funkcije kao setStarost.

Pa – već znate dosta toga: Kako da primite ulaz i vratite izlaznu promenljivu, da pravite strukture komplikovanih programa i da vršite razna računanja; Pa ipak najbolje tek dolazi.

Koje smo sastojke koristili do sada u našim programima? Brojeve i rečenice, zar ne? Nekako dosadno... Sad treba da upoznamo nekoliko sastojaka koji će učiniti programiranje zanimljivim.

Strukture podataka su sastojak koji strukturise podatke. (Ma vidi ti to...) Samostalan broj u stvari nema neku strukturu, zar ne? ali recimo da želimo da stavimo nekoliko brojeva zajedno u jedan sastojak – to bi imalo neku strukturu. Na primer, mozda želimo listu brojeva. To je lako:

```
[3,6,78,93]
```

Pominjali smo liste u odeljku o petljama, ali zapravo nismo rekli puno o njima. Pa – evo kako ih pravimo. Samo navedemo elemente odvojene zarezom i zatvorimo uglaste zagrade.

Da pogledamo primer koji računa proste brojeve (deljive samo sa sobom i sa 1)::

```
# Racuna sve proste brojeve ispod 1000
# (Ne bas najbolji nacin da se to obavi,ali...)

rezultat = []
kandidati = range(3,1000)
```

```
baza = 2
proizvod = baza

while kandidati:
....while proizvod < 1000:
........if proizvod in kandidati:
............kandidati.remove(proizvod)
........proizvod = proizvod+baza
....rezultat.append(baza)
....baza = kandidati[0]
....proizvod = baza
....del kandidati[0]

rezultat.append(baza)
print rezultat
```

Nove stvari u ovom primeru.

1. Interval brojeva ugradjen u funkciju može se koristiti kao bilo koji drugi interval.. (uključuje prvi broj, ali ne i poslednji.)
2. Lista (interval,raspon) se može koristiti kao logička promenljiva. Ako nije prazna, onda je tačna (true) , a ako jeste prazna onda je netačna (false). Tako, while kandidati znači "dok lista kandidata nije prazna" ili jednostavnije "dok još uvek ima kandidata".
3. Možete pisati IF neki element IN neka lista da proverite da li se on nalazi u njoj.
4. Možete napisati neka_lista.remove(neki_element) da izbacite taj element iz date liste.
5. Možete dodati jednu listu na drugu koristeći neka_lista.append(druga_lista). U stvari, možete koristiti i + (kao npr. jednalista = drugalista + nekatreca) ali to nije jako efikasno.
6. Možete doći do elementa liste zadajući njegovu poziciju kao broj (gde je prvi element, začudo, element 0) u uglastim zagradama iza naziva liste. Tako Nekalista[3] je četvrti element intervala nekalista. (Opsirnije o ovom kasnije.)
7. Možete da brišete promenljive koristeći ključnu reč del. Ona se može koristiti (kao ovde) za brisanje elemenata iz liste. Tako del nekaLista[0] briše prvi element iz nekaLista. Ako je lista bila [1,2,3] pre brisanja, biće [2,3] posle.

Pre nego što krenem sa objasnjenjem misterije indeksiranja liste elemenata, daću kratko objašnjenje prethodnog primera. Ovo je verzija prastarog algoritma po imenu "The Sieve of Erastothenes"(ili tako nešto). On podrazumeva paket (ili u ovom slučaju listu tj. niz) brojeva kandidata, i sistematski uklanja brojeve za koje zna da nisu prosti. Kako to zna? Zato što su oni proizvodi dva druga broja.

Počinjemo listom kandidata koja sadzi brojeve [2..999] – znamo da je 1 prost broj (u stvari, mozda i nije, zavisi koga pitate), i želimo sve proste brojeve manje od 1000. (U stvari naša lista je [3..999], ali i 2 je kandidat, jer nam je on prva baza). Takođe imamo i listu Rezultat koja sadrzi sve do tada dodate rezultate. Na početku, ova lista sadrzi samo broj 1. Takođe imamo promenljivu nazvanu baza. U svakom prolazu kroz algoritam, mi

uklanjamo sve brojeve koji su nekako deljivi sa ovim brojem koji je baza (koji je uvek najmanji mogući od svih kandidata). Posle svakog prolaza, znamo da je najmanji broj koji je preostao prost (jer su svi brojevi koji su bili proizvodi nekih manjih izbačeni – shvatate li?). Zbog toga, dodajemo ga nizu Rezultat, kao novu bazu postavljamo vrednost ovog broja, i sklanjamo ga sa liste kandidata (da ga ne obradjujemo ponovo.) Kada lista kandidata ostane prazna, u nizu Rezultat biće svi prosti brojevi. Pametno, a?

Stvari o kojima treba razmisliti:

Šta je specijalno kod prvog prolaza? Ovde je baza 2, pa ipak i ona je uklonjena u izbacivanju? Zašto? Zašto se to ne dogodi sa drugim bazama? Možemo li biti sigurni da je proizvod uvek u listi kandidata kada želimo da ih uklonimo? Zašto?

Sad – šta je sledeće? Ah, da ... Indeksiranje i isecanje. Ovo je način da dodjemo do samostalnih elemenata iz Python-ovih lista. Već ste videli kako funkcionise obično indeksiranje. Prilično je linearno. U stvari, rekli smo sve što treba da znate o tome, osim jedne stvari : negativna oznaka znači da se računa od kraja liste. Znači, nekaLista[-1] je poslednji element iz niza nekaLista, nekaLista[-2]je pretposlednji i tako dalje.

6. Objekti i objektno-orijentisano programiranje

Evo jedne, ako takva postoji, veoma zvučne reči : "Objektno-orijentisano programiranje."

kao što naslov poglavlja sugeriše, ova vrsta programiranja je samo još jedan način skraćivanja posla i ne petljanja sa detaljima. Procedure skraćuju kompleksne operacije tako što im daju ime koje se kasnije poziva u programu. U OOP, ne tretiramo na ovaj način samo operacije, već i objekte. (E sad, to mora da je bilo veliko iznenadjenje, a?) uzmimo za primer, da želimo da napravimo program za kuvanje preliva, umesto da pišemo mnogo procedura koje se bakću sa temperaturom, vremenom, sastojcima i tako dalje, mogli bi da ih strpamo sve zajedno u preliv-objekt. Ili bi mozda mogli da imamo i šerpa-objekt i vreme-objekt ... Sada, stvari kao što je temperatura bile bi samo osobine preliv-objekta, dok bi se vreme moglo očitavati iz sat-objekta. A da naš program radi nešto, mogli bi da naučimo naše objekte neke metode; na primer, šerpa-objekt bi mogla da zna kako da skuva preliv itd.

Pa, kako to radimo u Python-u? Ne možemo direktno napraviti objekt. Umesto pravljenja šerpe, napravićemo 'recept' koji govori kako se pravi šerpa, napravićemo uputstvo koje će opisati šerpu. ovo uputstvo opisuje klasu (class) objekta koji zovemo šerpa. Veoma jednostavna klasa za šerpu bi bila:

```
class Serpa:
....def insertPreliv(self, preliv):
.......self.preliv = preliv
....def getPreliv(self):
........return self.preliv
```

Jel' ovo izgleda čudno?

Nove stvari u ovom primeru...

1. Klase objekata definisu se ključnom reci class.
2. Imena klasa obično počinju velikim slovom, dok funkcije i promenljive (pa i metode i osobine) počinju malim slovom.
3. Metode (tj. funkcije i operacije koje objekti znaju da obave) su definisane na standardan način, ali unutar bloka klase.
4. Sve metode objekata treba da imaju prvi parametar koji se zove self (ili nešto slično...) Razlog će (nadam se) postati jasan uskoro.
5. Osobinama i metodama objekata se pristupa ovako: mojPreliv.temperatura = 2, ili debil.budi_dobar().

Mogu da pogodim da su vam neke stvari u vezi primera još uvek nejasne. na primer, šta je ta self stvar? A sada kad imamo recept za objekt (tj. njegovu klasu), kako pravimo sam objekt?

Prihvatimo se prvo ovog poslednjeg. Objekt se stvara pozivanjem imena klase kao da je funkcija

```
mojaSerpa = Serpa()
```

mojaSerpa sada sadrzi serpa objekt, često zvan primer klase Serpa. Pretpostavimo da smo napravili i klasu Preliv; tada bi mogli da uradimo nešto kao:

```
mojPreliv = preliv()

mojaSerpa.insertPreliv(mojPreliv)
```

mojaSerpa.preliv bi sada sadržala mojPreliv. Kako to? Zato što, kada pozovemo jednu metodu objekta, prvi parametar, obično nazvan self, uvek sadrzi sam objekt. (pametno, a?) zato, linija self.preliv = preliv postavlja osobinu preliv trenutnog objekta serpa na vrednost parametra preliv. uočite da su ovo dve različite stvari, iako se obe zovu preliv u ovom primeru.

Obzirom na činjenicu da našim kursom nije predviđeno ozbiljnije izučavanje i primena Pythona, ovde ćemo se zaustaviti. Nadamo se da Vam je ovaj tutorial pomogao da steknete osnovna znanja o Python programskom jeziku.