Zagrljaj

Pitolia

Swaroop CH

Prevod:

BagZačin

Sadržaj

1 Zagrljaj Pitona

"Zagrljaj Pitona" je besplatna knjiga o programiranju korišćenjem Pajton (Python) programskog jezika. Ona služi kao uvod ili vodič za Python jezik početnicima u programiranju. Ako je sve što znate da radite na kompjuteru čuvanje tekstualnih datoteka, onda je ovo knjiga za Vas.
Ova knjiga je pisana za najnoviju verziju Python 3, iako je Python 2 najčešće korišćena verzija Python-a danas (pročitajte više o tome u Razlike između verzija Python 2 i Python 3).

1.1 Ko čita "Zagrljaj Pitona"?

Evo šta ljudi govore o knjizi:
"Najbolja stvar na koju sam naleteo je "Zagrljaj Pitona", koji je jednostavno brilijantna knjiga za početnika. Ona je dobro napisana, pojmovi su dobro objašnjeni pomoću očiglednih primera."
- *Syed Talal* (19 godina)
"Ovo je najbolji početnički tutorijal koji sam ikada video! Hvala vam na trudu."
- *Walt Michalik* (wmich50@theramp.net)
"Napravili ste najbolji Python tutorial koji sam našao na internetu. Odličan posao. Hvala!"
- *Joshua Robin* (joshrob@poczsta.onet.pl)
"Odličan i pažljiv uvod u programiranje # Python za početnike "
- Shan Rajasekaran
"Zdravo, ja sam iz Dominikanske Republike. Moje ime je Pavel, nedavno sam pročitao vašu knjigu "Zagrljaj Pitona" i ja je smatram fantastičnom! :). Naučio sam mnogo iz primera. Vaša knjiga je od velike pomoći za početnike kao što sam ja ... "
- *Pavel Simo* (pavel.simo@gmail.com)
"Nedavno sam završio čitanje "Zagrljaj Pitona", i mislio sam da stvarno treba da ti se zahvalim. Veoma mi je žao što posle poslednje stranice ove knjige moram da se vratim na dosadne i suvoparne O'reilly i slične priručnika za učenje Python-a. U svakom slučaju, ja stvarno cenim tvoju knjigu."
- *Samuel Young* (sy137@gmail.com)
"Dragi Swaroop, ja pohađam časove kod profesora koji nema interesovanje za učenje. Mi koristimo "Learning Python", drugo izdanje, od O'Reilly-ja. To nije knjiga za početnika bez ikakvog programerskog znanja, i profesora koji bi trebalo da radi nešto drugo. Hvala vam puno za Vašu knjigu, bez koje bi bio izgubljen u Python-u i programiranju. Hvala Vam još milion puta, vi ste u stanju da "razložite problem u komade" na nivo koji može da razume i početnik, a to ne može svako."
- *Joseph Duarte* (jduarte1@cfl.rr.com)
"Ja volim tvoju knjigu! Ona je najbolji Python tutorijal ikada napisan, i veoma je korisna referenca. Brilijantno, remek delo! Samo tako nastavite sa dobrim radom!"
- *Chris-Andrė Sommerseth*
"Samo da Vam pošaljem e-mail kako bih vam se zahvalio na knjizi "Zagrljaj Pitona". Ja sam pokušavao da naučim Python nekoliko meseci pre pronalaska Vaše knjige, mada sam postigao ograničen uspeh koristeći pyGame, nisam uspešno napravio niti jedan program.
Zahvaljujući Vašem pojednostavljenju problema, Python sada izgleda kao lako dostupan cilj. Izgleda da sam konačno naučio osnove pa sad mogu nastaviti ka svom početnom cilju: razvoj video igrica.
...
Još jednom, hvala PUNO za stvaranje ovog strukturisanog i korisnog vodiča za osnove programiranja na internetu. On mi je pokazao načine "u" i "iz" OOP-a, jednostavno i sa

razumevanjem – mesta na kojima su pokušaji druge dve tekstualne knjige propali."
- *Matt Gallivan* (m_gallivan12@hotmail.com)
"Želeo bih da vam se zahvalim na vašoj knjigzi "Zagrljaj Pitona" za koju ja mislim da je najbolji način da se nauči Python. Ja imam 15 godina i živi u Egiptu. Moje ime je Ahmed. Python je bio moj drugi programski jezik Učim Visual Basic 6 u školi, ali u njemu ne uživam, ali sam zaista uživao uz učenje Python-a. Uspešno sam napravio adresar program iz primera. Ja ću pokušati da napravim što više programa uz čitanje "izvora" Python programa (ako možete da mi kažete izvore koji će biti od pomoći). Takođe ću početi da učim Javu i ako možete da mi kažete gde da nađem tutorijal koji je dobar kao Vaš, samo pisan za Javu. To bi mi mnogo pomoglo. Hvala."
- *Ahmed Mohammed* (sedo_91@hotmail.com)
"Divan resurs za početnike koji žele da nauče više o Python-y, koji je na 110 stranica PDF-a! Tutorijal "Zagrljaj Pitona" od Swaroop C H-a je odlično napisan, lako se prati, i možda je najbolji uvod u Python programiranje koji je trenutno dostupan."
- *Drew Ames* u članku Scripting Scribus objavljen na linux.com
"Juče sam prelistao veći deo "Zagrljaj Pitona" na mojoj Nokia N800 i mislim da je to najlakši i najprecizniji uvod u Python na koji sam naišao. Preporučujem kao polaznu tačku za učenje Python-a."
- *Jason Delport* na svom blogu.
"Zagrljaj Vim-a i Pitona (A Byte of VIM, A Byte Of Python – originalni naziv) od @swaroopch je, po meni, daleko najbolji rad u sferi tehničkog pisanja. Predivno štivo # FeelGoodFactor"
- *Surendran* u svom tveet-u.
""Zagrljaj Pitona" je daleko najbolja stvar za čoveka."
(U odgovoru na pitanje: "Može li iko predložiti dobar i jeftin resurs za učenje osnove Python-a?")
- *Justin LoveTrue* kaže na Facebook stranici zajednice
"Knjiga "Zagrljaj Pitona" je bila veoma korisna .. Hvala na trudu :)"
- Chinmay
"Uvek sam bio ljubitelj "Zagrljaj Pitona" jer je napravljen kako za nove tako i za iskusne programere."
- Patrick Harrington, u svom StackOverflow odgovoru.
Čak i NASA! Ovu knjigu čak koristi NASA! Ona se koristi u njihovoj Jet Propulsion Laboratory u njihovom projektu "Deep Space Network Project".

1.2 Akademska predavanja

Ova knjiga se (ili se još koristi) koristila kao nastavni materijal u različitim obrazovnim institucijama:

- "Principles of Programming Languages" predavanje na Vrije Universiteit, Amsterdam
- "Basic Concepts of Computing" predavanje na University of California, Davis
- "Programming With Python" predavanje na Harvard University
- "Introduction to Programming" predavanje na University of Leeds
- "Introduction to Application Programming" predavanje na Boston University
- "Information Technology Skills for Meteorology" kurs na University of Oklahoma
- "Geoprocessing" predavanje na Michigan State University
- "Multi Agent Semantic Web Systems" predavanje na University of Edinburgh

1.3 Licenca

Što znači:

- Vi ste slobodni da je delite, odnosno da kopirate, distribuirate i dajete drugima ovu knjigu.
- Vi ste slobodni da je menjate i transformišete odnosno da prilagodite ovu knjigu.
- Vi ste slobodni da je koriste u komercijalne svrhe.

Ali imajte na umu:

- Molimo vas da ne prodajete elektronske i štampane primerke ove knjige, osim ako je jasno i uočljivo u opisu naznačeno da to nije od originalnog autora ove knjige.
- Naznake o originalnom radu moraju biti prikazani u uvodnom opisu knjige i na naslovnoj strani dokumenta, linkovanjem nazad na http://www.swaroopch.com/notes/Python i mora jasno da se ukaže da originalni tekst može biti preuzet sa ove lokacije.
- Svi kodovi/skripte prikazane u ovoj knjizi su licencirane pod 3-clause BSD licence osim ako nije drugačije naznačeno.

1.4 Čitajte je sad

Možete da pročitate knjigu na internetu . (original na Engleskom jeziku).

1.5 Kupite Knjigu

Štampana verzija knjige se može kupiti zarad zadovoljstva čitanja, kao i kako bi podržala dalji razvoj i unapređenje ove knjige.

1.6 Preuzimanje

- PDF
- Ceo izvor

Ako želite da podrži nastavak razvoja ove knjige, razmislite o kupovini papirne verzije.
(linkovi su ka originalnim verzijama na Emgleskom jeziku).

1.7 Čitajte knjigu na vašem maternjem jeziku

Ukoliko ste zainteresovani za čitanje ili doprinos prevodu ove knjige na druge svetske jezika, pogledajte stranice prevoda.

2 Predgovor

Python je verovatno jedan od retkih programskih jezika koji je i jednostavan i moćan. On je dobar kako za početnike, tako i za stručnjake, i još važnije, zabavno je programiranje sa njim. Ova knjiga ima za cilj da pomogne da naučite ovaj divan jezik i pokaže kako da se stvari urade brzo i bezbolno - u suštini “Savršena Anti-glavobolja za vaše programske probleme”.

2.1 Za koga je pisana ova knjiga

Ova knjiga služi kao vodič ili tutorijal kroz Python programski jezik. Ona je uglavnom usmerena ka početnicima ali je korisna i za iskusne programere.
Cilja na to da ako je sve što znate o kompjuterima je kako da sačuvate tekstualnu datoteku, onda možete da naučite i Python iz ove knjige. Ukoliko imate prethodno iskustvo iz programiranja, onda takođe možete naučiti Python.
Ako imate prethodno iskustvo iz programiranja, bićete zainteresovani za razlike između Python-a i vašeg omiljenog programskog jezika - pa su naglašene mnoge takve razlike. Nešto ipak treba da Vas brine: Python će uskoro postati vaš omiljeni programski jezik!

2.2 Lekcija iz istorije

Prvi put sam se susreo sa Python-om kada sam trebao da isprogramiram instaler softvera koji sam pisao, nazvan “Diamond”, tako da bi mogao da ga načinim da se instalira lako. Morao sam da biram između Python-ovih i Perl-ovih povezivanja sa Qt bibliotekama. Uradio sam neka istraživanja na internetu i naišao sam na članak od Eric S. Raymond-a, koji je poznat i cenjen haker, u kome je pisao o tome kako je Python postao njegov omiljeni programski jezik. Takođe sam saznao da su PyQt vezovi bili "u boljem stanju" u odnosu na Perl-Qt. Dakle, odlučio sam da je Python jezik za mene.
Onda sam pošao u potragu za dobrom knjigom za Python. Nisam mogao da nađem niti jednu! Pronašao sam neke O’Reilly-jeve knjige, ali su one ili bile preskupe ili su bili više kao referentni priručnik, nego vodič za programiranje. Zato sam ja proučavao dokumentaciju koju sam dobo uz Python. Međutim ta dokumentacija je bila suviše kratka i “suvoparna”. Ona mi je ipak dala dobru osnovu o Python-u, ali ta osnova nije bila potpuna. Uspeo sam da uradim nešto sa jezikom, jer sam imao prethodno iskustvo programiranja, ali njegova dokumentacija nije pogodna za početnike.
Oko šest meseci posle mog prvog susreta sa Python-om sam instalirao (tada) najnoviji Red Hat 9.0 Linux pa sam se malo igrao sa KWord-om. Toliko sam bio oduševljen time, da mi je odjednom sinula ideja da napišem neke stvari o Python-u. Počeo sam da pišem nekoliko stranica, ali se ubrzo nakupilo 30 strana. Onda sam počeo ozbiljno da razmišljam o tome, da bi sve to bilo više korisno u obliku knjige. Posle *dosta* ispravljanja grešaka, sve to je dostiglo fazu u kojoj je postalo koristan vodič za učenje Python jezika. Smatram da će ova knjiga biti moj doprinos i promocija open source zajednice.
Ova knjiga je počela kao mojia lična beleška o Python-u, i ja je i dalje posmatram na isti način, iako sam uložio mnogo napora da bi bila čitljivija drugima :)
U istinskom duhu otvorenog koda, ja sam dobio mnogo konstruktivnih predloga, kritika i povratnih informacija od oduševljenih čitalaca koji su mi pomogli mnogo u poboljšanju ove knjige.

2.3 Status knjige

Ova knjiga je formatirana u oktobru 2012 korišćenjem Pandoc-a, kako bi se mogla generisati u formatu elektonske knjige, a sve to na zahtev više korisnika, kada su urađene i ispravke i dopune.
Promene u izdanju iz decembra 2008-me (sem ranije velike revizije u martu 2005) je ažuriranje za

izdanje Python 3.0 jezika.
Ovoj knjizi je potrebna pomoć svojih čitalaca, kakva bi bila da i sami istaknu one delove knjige koji nisu dobro napisani, nisu razumljivi ili su jednostavno pogrešni. Molimo pišite glavnom autoru ili odgovarajućim prevodiocima sa vašim komentarima i sugestijama.

2.4 Zvanična internet stranica

Zvanična internet stranica knjige je http://www.swaroopch.com/notes/Python. Na njoj možete pročitati celu knjigu, preuzeti najnovije verzije knjige, kupiti štampanu verziju, a takođe i poslati mi poruku sa Vašom povratnom informacijom (utisci, sugestije, predlozi...).

2.5 Par rečenica za razmišljanje

"Postoje dva prilaza u izgradnji i dizajnu softvera: jedan način je da ga načinite tako jednostavno da očigledno nema nedostataka, druga je da ga napravite toliko komplikovano da nema očiglednih nedostataka."
- C. A. R. Hoare
"Za uspeh u životu nije bitan toliko talenat i mogućnost, koliko koncentracija i upornost."
- C. W. Wendte

3 Uvod

Python je jedan od onih retkih jezika za koji može da se kaže da je i *jednostavan* ali i *moćan*.
Prijatno ćete se iznenaditi kada vidite kako je lako da se koncentrišete na rešavanje problema umesto na sintaksu i strukturu jezika sa kojem programirate.
Zvanični uvod u Python je:
"Python je jezik koji se lako nauči,a u isto vreme je i moćan programski jezik. On ima efikasne high-level strukture podataka i jednostavan ali efikasan pristup objektno-orijentisanom programiranju. Python-ova elegantna sintaksa i dinamika, zajedno sa svojim prirodom interpretera, čine ga idealnim jezikom za pisanje skripti i brz razvoj aplikacija u mnogim oblastima i na većini operativnih sistema."
Ja ću pojasniti većinu ovih funkcija detaljnije u narednom odeljku.
Priča iza imena jezika
Guido van Rossum, tvorac Python jezika, nazvao je jezik po BBC emisiji "Leteći Cirkus Monti Pajtona" (“Monty Python’s Flying Circus”). On nije bio ljubitelj zmija koja ubijaju životinje (svoju hranu) tako što obavijaju svoja duga tela oko žrtve i tako ih smrskaju.

3.1 Karakteristike Python-a

3.1.1 Jednostavanost

Python je jednostavan i minimalističan jezik. Čitanje dobrog izvornog koda Python programa budi osećaj kao čitanje engleskog jezika, iako je to veoma striktan Engleski! Ovakva priroda Python-a je jedan od njegovih najvećih prednosti. Ona vam omogućava da se više koncentrišete na rešavanje problema nego na sama pravila programskog jezika.

3.1.2 Lako se uči

Kao što ćete videti, sa Python-om je izuzetno lako da se počne sa programiranjem. Python ima izuzetno jednostavnu sintaksu, kao što je već pomenuto.

3.1.3 Beaplatan je i otvorenog koda (Free and Open Source)

Python je primer *FLOSS* (Free/Libré and Open Source Software) softvera. Jednostavnije rečeno, možete slobodno distribuirati kopije ovog softvera, čitati izvorni kod, napraviti promene na njemu, i koristiti njegove delove u novim slobodnim programima. FLOSS se zasniva na konceptu zajednice koja deli znanje. Ovo je jedan od razloga zašto je Python tako dobar - jer je stvaran i stalno poboljšavan od strane zajednice koji samo želi da stvori bolji Python.

3.1.4 High-level Jezik

Kada pišete programe u Python-u, nikada nećete morati da se bavite o low-level detaljima kao što su upravljanje memorijom koju koristi Vaš program, itd.

3.1.5 Prenosiv (Portable)

Zbog svoje prirode otvorenog koda, Python je portovan (tj. izmenjen da bi radio) na više platformi. Svi vaši Python programi mogu raditi na bilo kojoj od ovih platformi, bez potrebe bilo kakve promene u kodu ako ste bili dovoljno pažljivi da izbegnete bilo kakve funkcije koje zavise od operativnog sistema.

Možete koristiti Python na Linux-u, Windows-u, FreeBSD-u, Macintosh-u, Solaris-u, OS/2, Amigi, AROS-u, AS/400, BeOS-u, OS/390, z/OS-u, Palm OS-u, QNX, VMS-u, Psion-u, Acorn RISC OS-u, VxWorks-u, PlayStation-u, Sharp Zaurus-u, Windows CE, pa čak i na PocketPC-u!
Možete čak koristiti platformu kao što je Kivy da stvarate igre za iOS (iPhone, iPad) i Android.

3.1.6 Interpreter

Ovo zahteva malo objašnjenje.
Program napisan na jezicima koji zahtevaju kompajliranje kao što su C ili C++ su konvertovani iz izvornog jezika (odnosno C-a ili C++-a) na jezik kojim govori Vaš računar (binarni kod tj. nule i jedinice) koristeći kompajler sa različitim oznakama i opcijama. Kada pokrenete takav program, linker/loader softver kopira program sa hard diska u memoriju i tada ga pokreće.
Python-u, sa druge strane, ne treba kompilacija u binarni kod. Vi samo pokrećete program direktno iz izvornog koda. U suštini, Python pretvara izvorni kod u neki srednji oblik, koji se zove bytecode (bajtkod) a zatim prevodi ovo u jezik računara koji ga pokreće. Sve ovo, u suštini, čini korišćenje Python-a mnogo lakšim, jer ne morate da brinete o kompajliranju programa, niti da vodite računa da li su odgovarajuće biblioteke povezane i učitane itd. Ovo takođe znači da je Vaš Python program prenosiv sa platforme na platformu, jer možete samo da kopirate Vaš Python program na drugi računar i on će da radi!

3.1.7 Objektno orijentisan

Python jednako podržava proceduralno-orijentisano programiranje, kao i objektno-orijentisano programiranje. U *proceduralno-orijentisanim* jezicima, program je izgrađen iz procedura ili funkcija koje nisu ništa drugo do komade programa koje se mogu više puta upotrebljavati. U *objektno-orijentisanim jezicima*, program je izgrađen od objekata koji kombinuju podatke i funkcionalnost. Python ima veoma moćan, ali pojednostavljen način obavljanja OOP-a, naročito kada se uporedi sa velikim jezicima kao što su C++ ili Java.

3.1.8 Proširiv

Ako vam je potrebno da važan deo koda mora da se pokreća veoma brzo, ili želite da imate neki deo algoritma da ne bude otvoren (da ne može da se pročita izvorni kod), možete kodirati taj deo vašeg programa u C-u ili C++-u, a zatim ga koristiti iz Python programa.

3.1.9 "Ugradljiv" u druge programe

Možete da "ugradite" Python u vaše C/C++ programe da bi pružili mogućnosti "skriptovanja" za korisnike vašeg programa.

3.1.10 Proširive biblioteke

Standardne biblioteke Python-a su zaista ogromne. One mogu da Vam pomognu da uradite razne stvari uključujući regular expressions, generisanje dokumenata, testiranje uređaja, threading-a, baze podataka, WEB pretraživača, CGI-a, FTP-a, e-mail-a, XML-a, XML-RPC-a, HTML-a, WAV fajlova, kriptografije, GUI (grafičkih korisničkih interfejsa), i drugih stvari koje su zavisne od sistema. Zapamtite, to sve je uvek na raspolaganju kada je Python instaliran. To je takozvana "*Batteries Included*" (spreman za rad-sve dolazi sa njim) filozofija Python-a.
Pored standardnih biblioteka, postoje razne druge kvalitetne biblioteke koje možete pronaći na Python Package Index internet strani.

Rezime
Python je zaista uzbudljiv i moćan jezik. On sadrži pravu kombinaciju performansi i karakteristika koje čine pisanje programa u Python-u zabavnim i lakim.

3.2 Razlike između verzija Python 2 i Python 3

Možete da preskočite ovu sekciju, ako niste zainteresovani za razlike između Python 2 i 3. Ali, molim vas, obratite pažnju na to koju verziju koristite.
Ova knjiga je prilagođena 2008g. za Python 3 i ona je bila jedna od prvih knjiga za one koji koriste Python 3. To je, nažalost, zbunjivalo čitaoce koji su pokušavali da nauče Python 2 sa verzijom knjige za Python 3 i obrnuto. Svet lagano i dalje prelazi na Python 3.
Dakle, vi ćete učiti da koriste Python 3 iz ove knjige, čak i ako želite da koristite Python 2. *Imajte na umu da kada ste pravilno razumeli i naučili da koriste bilo koji od njih, možete lako da naučite i razlike između te dve verzije i da se prilagodite vrlo lako. Najteži deo je učenje programiranja i razumevanje osnove Python jezika i to je naš cilj u ovoj knjizi, i kada postignete taj cilj, lako možete da koristite kako verziju 2 tako i verziju 3 u zavisnosti od situacije u kojoj se nađete.*
Za detalje o razlikama između Python 2 i Python 3 verzije, pogledajte ovu internet stranicu na Ubuntu wiki-ju.

3.3 Šta programeri kažu?

Možda će vam biti interesantno da pročitam šta veliki hakeri poput ESR imaju da kažu o Python-u:

1. *Eric S. Raymond* je autor knjige "The Cathedral and the Bazaar", a takođe je i osoba koja je izmislila termin *Open Source* (otvoreni kod). On kaže da je Python postao njegov omiljen programski jezik. Ovaj članak je bio prava inspiracija za moje prve korake sa Python-om.
2. *Bruce Eckel* je autor čuvenih knjiga *Thinking in Java* i *Thinking in C++*. On kaže da ga nijedan jezik nije učinio više produktivnijim od Python-a. On kaže da je Python možda jedini jezik koji se fokusira da pravljenje stvari bude lakše za programera. Za više detalja pročitajte kompletan intervju.
3. *Peter Norvig* je poznati autor LISP-a, i direktor za Google-ovog Search Quality (zahvaljujem Guido van Rossum-u za tu informaciju). On kaže da je Python oduvek bio sastavni deo Google-a. Možete se zapravo i uveriti sami gledajući Google Jobs stranu na internetu u kojoj se navodi znanje Python-a kao uslov za poslove softverskih inženjera.

4 Instalacija

U ovom poglavlju ćemo se posvetiti načinu na koji se instalira Python programski jezik, na više različitih operativnih sistema. Nemojte ga preskakati jer sadrži dosta korisnih informacija.

4.1 Instalacija na Windows-u

Posetite sajt http://www.python.org/download/ i preuzmite najnoviju verziju. Instalacija je ista kao i kod svakog drugog Windows baziranog programa.
Pažnja!
Kada se prilikom instalacije prikaže mogućnost "deštikliranja" nekih "opcionih" komponenti, nemojte to uraditi - ostavite onako kako Vam je ponuđeno.

4.1.1 DOS komandna linija

Ako želite da imate mogućnosti da koristite Python u Windows komandnoj liniji, odnosno DOS prompt-u, onda morate da na odgovarajući način podesite promenljivu putanje PATH.
Za Windows 2000, XP, 2003: kliknite na `Control Panel - System - Advanced - Environment Variables`. Kliknite na promenljivu pod imenom PATH u odeljku "System Variables", a zatim izaberite `Edit` i dodajte `;C:\Python33` (proverite da li ova fascikla postoji, može biti drugačiji za novije verzije Python-a) na kraju liste onoga što se već tamo nalazi. Naravno, koristite odgovarajuće ime direktorijuma u kojem je instaliran Python.
Za starije verzije operativnog sistema Windows, otvorite datoteku `C:\AUTOEXEC.BAT` i dodajte liniju `"PATH=%PATH%;C:\Python33"` (bez navodnika) i restartujte sistem. Za Windows NT, koristite `AUTOEXEC.NT` fajl.
Za Windows Vista-u:

1. Kliknite na dugme Start i izaberite stavku Control Panel
2. Kliknite System, sa desne strane ćete videti "View basic information about your computer"
3. Sa leve strane je lista zadataka, od kojih je poslednja "Advanced system settings".Kliknite na nju.
4. Prikazana Vam je Advanced kartica dijaloga System Properties. Kliknite na dugme Environment Variables koje se nalazi u donjem desnom uglu.
5. U donjem polju pod nazivom "System Variables" skrolujte prema dole do Path i kliknite na dugme Edit.
6. Promenite path prema direktorijumu u koji ste instalirali Python.
7. Ponovo pokrenite sistem. Vista neće podesiti putanje u sistemu dok se ne restartuje.

Za operativni sistem Windows 7:

1. Desnim tasterom miša kliknite na ikonicu kompjutera na Vašem desktopu i izaberite properties ili kliknite na Start dugme i izaberite Control Panel — System and Security — System. Kliknite na Advanced system settings sa leve strane, a zatim kliknite na tab Advanced. Na dnu kliknite na Environment Variables i pod System variables, potražite promenljivu PATH, izaberite je i pritisnite Edit.
2. Na kraju linije ispod Variable value dodati `;C:\Python33` .
3. Ako je vrednost bila `%SystemRoot%\system32;` ona bi trebala da postane `%SystemRoot%\system32;C:\Python33`
4. Kliknite na OK i završili ste. Restart nije potreban.

4.1.2 Pokretanje Python konzole na Windows-u

Korisnici Windows-a, mogu da pokreću Python interpreter u komandnoj liniji, ako su postavili PATH promenljivu sa odgovarajućim parametrima.
Da biste otvorili terminal u operativnom sistemu Windows, kliknite na dugme Start i kliknite na "Run". U dijalogu otkucajte `cmd` i pritisnite Enter.
Zatim, odkucajte `python3 -V` (ukoliko ova komanda pokazuje neku grešku ili ne pokazuje ništa, a prethodni postupak ste dobro uradili, pokušajte sa komandom `python -V`, ukoliko ni ona "ne radi", proverite prethodne korake, najverovatnije niste dobro podesili PATH promenljivu) da bi bili sigurni da nema grešaka.

4.2 Instalacija na Mac OS X-u

Za Mac OS X korisnike: otvorite terminal pritiskom kombinacije `Command+Space` tastera (otvara Spotlight pretragu), kucajte `Terminal` i pritisnite Enter.
Instalirajte Homebrew kucanjem:

```
ruby -e "$(curl -fsSkL raw.github.com/mxcl/homebrew/go)"
```

Zatim instalirajte Python 3 kucajući:

```
brew install python3
```

Sada, kucajte `python3 -V` da proverite da nema grešaka.

4.3 Instalacija na Linux-u

Za Linux korisnike: otvorite terminal otvaranjem `Terminal` aplikacije ili pritiskom na Alt + F2 i kucanjem `gnome-terminal` (ako koristite GNOME desktop okruženje, ako ne, ukucajte komandu koja pokreće odgovarajući terminal emulator). Ako to ne uspe, pogledajte dokumentaciju ili se obratite na forumima vaše Linux distribucije.
Zatim, morate da instalirate python3 paket. Na primer u Ubuntu, možete da koristite `sudo apt-get install python3` . Molimo proverite dokumentaciju ili forume Vaše Linux distribucije koju imate instaliranu za ispravanu komandu za pokretanje paket menadžera distribucije.
Kada ste završili instalaciju, pokrenite `python3 -V` u šelu i trebalo bi da vidite verziju Python-a na ekranu:

```
$ python3 -V
Python 3.3.0
```

Pažnja:
$ je deo šela. To može biti drugačije kod Vas u zavisnosti od podešavanja Vašeg operativnog sistema na računaru, ovde ćemo navoditi odziv šela sa $ simbolom.
Možda je podrazumevan u novoj verziji Vaše distribucije?
Novije distribucije, kao što je npr Ubuntu 12.10, dolaze sa već instaliranim Python 3, kao podrazumevanom verzijom Python-a, pa proverite da nije možda već instaliran.

4.4 Rezime

Od sada, smatramo da imate Python 3 instaliran na vašem sistemu.
U sledećem poglavlju ćemo krenuti sa pisanjem našeg prvog Python 3 programa.

5 Prvi koraci

Sada ćemo videti kako da pišete i pokrenete tradicionalno prvi program "Hello World" ("Pozdrav Svetu") u Python-u. To će vas naučiti kako da pišete, sačuvate i pokrećete Python programe.
Postoje dva načina "naterati" Python da pokreće Vaš program - pomoću interaktivnog prompt-a ili korišćenjem izvornog fajla. Sada ćemo videti kako da koriste oba metoda.

5.1 Korišćenje interaktivnog prompt-a

Otvorite terminal u vašem operativnom sistemu (kao što je objašnjeno ranije u poglavlju koje govori o instalaciji), a zatim otvorite Python prompt kucanjem `python3` (ili samo `python`, ukoliko se program "odaziva" na taj način. PAŽNJA! Budite sigurni da je verzija programa 3! To ćete proveriti sa komandom iz prethodnog poglavlja. Po standardu označavanja `python` je komanda za verziju 2, a `python3` za verziju 3, ali npr 3.3.0 na OS WindowsXP se poziva samo naredbom `python`) i pritiskom na taster Enter.
Kada ste pokrenuli `python3` , trebalo bi da vidite ovakvu oznaku: `>>>` . Odatle možete početi da kucate. To se zove *Python interpreter prompt* (interaktivni prompt).
U Python interaktivnom promptu, ukucajte `print('Pozdrav Svima!')` zatim stisnite Enter. Trebalo bi da vidite reči `Pozdrav Svima!` na Vašem ekranu.
Evo ga primer šta bi trebalo da se vidi, ukoliko se koristi računar sa Mac OS X-om. Detalji o Python programu će se razlikovati u zavisnosti od Vašeg računara, ali deo od prompta (tj. od >>> nadalje) trebalo bi da bude isti bez obzira na operativni sistem.

```
$ python3
Python 3.3.0 (default, Oct 22 2012, 12:20:36)
[GCC 4.2.1 Compatible Apple Clang 4.0 ((tags/Apple/clang-421.0.60))] on
darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Pozdrav Svima!')
Pozdrav Svima!
>>>
```

Primetimo da Vam Python gotovo trenutno daje izlaz komande! Ono što ste upravo ukucali je jedna *komanda* u Python-u (Python statement). Mi koristimo komandu `print` da (kakvo iznenađenje?), prikažemo nekakvu vrednost koju smo zadali u komandi. Ovde smo zadali tekst `Pozdrav Svima!` i on je odmah prikazan na ekranu.
Kako izaći iz interaktivnog prompta
Ako koristite Linux ili Unix shell, možete da izađete iz interaktivnog prompta pritiskom ctrl+d ili unošenjem `exit()` (Napomena: Ne zaboravite da otkucate zagrade, '()'), zatim pritisnite enter. Ako koristite Windows komandnu liniju, pritisnite ctrl+z, a zatim enter taster.

5.2 Izbor uređivača teksta (Editora)

Ne možemo da pišemo naš program u interaktivni prompt svaki put kada želimo da ga pokrenemo, tako da bi bilo pametno da ih sačuvamo u datotekama kako bi mogli da ih pokrećemo neograničen broj puta.
Da biste kreirali Vaše Python izvorne fajlove, potreban nam je program (tekstualni editor) u kojem možete da kucate i sačuvate. Dobar programerski editor će učiniti Vaš život lakšim dok pišete izvorne datoteke. Dakle, zaista, izbor editora je od ključnog značaja. Morate da izaberete editor kao što bi odabrali auto koji bi kupili za Vaše potrebe. Dobar editor će vam pomoći da lakše pišete

Python programe, čineći vaše putovanje udobnijim i pomaže Vam da stignete na odredište (da postignete Vaš cilj) na mnogo brži i sigurniji način.
Jedan od osnovnih zahteva koji treba da ispuni Vaš editor je *isticanje sintakse*, to jest, bojenje različitih delova vašeg Python programa, tako da možete vizuelno da pratite svoj program i vizuelno zamislite njegov tok.
Ukoliko nemate ideju sa kojim editorom da počnete, ja bih preporučio korišćenje programa Komodo Edit, koji je dostupan za Windows, Mac OS X i Linux.
Ako koristite Windows, molim Vas da **ne koristite Notepad** - jer je loš izbor, jer ne postoji isticanje sintakse i, što je najvažnije - ne podržava uvlačenje teksta (indentaciju) što je veoma važno u našem učenju, kao što ćemo videti kasnije. Dobri editori poput Komodo Edit-a će sve to automatski raditi za nas.
Ako ste iskusan programer, onda mora da već koristite Vim ili Emacs. Izlišno je reći da su ovo dva najmoćnija editora i da ćete imati ogromne koristi upotrebljavajući ih u pisanju Python programa. Ja lično koristim oba za većinu mojih programa, pa sam čak i napisao celu knjigu o Vim-u. U slučaju da ste spremni da odvojite malo vremena da nauče Vim ili Emacs, onda Vam preporučujem da naučite da koristite bilo koji od njih, što će biti veoma korisno za Vas na duže staze. Međutim, kao što sam ranije pomenuo, početnici mogu početi sa Komodo Edit-om da bi se fokusirali na učenje o Python-u umesto na učenje upotrebe editora.
Da podsetimo, molimo Vas da izaberete odgovarajuću editor - to može da učini pisanje Python programa zabavanim i jednostavnim.
Za korisnike Vim-a
Postoji dobar članak o tome kako da podesite Vim da postane moćan Python IDE od John M Anderson-a. Takođe, preporučuje se jedi-vim plugin i moja sopstvena dotvim konfiguracija.
Za Emacs korisnike
Postoji dobar uvod o tome kako da napravite od Emacs-a moćni Python IDE od Pedro Kroger-a. Takođe, preporučujem BG's dotemacs konfiguraciju.

5.3 Korišćenje izvorne datoteke

Hajde sad da se vratimo na programiranje. Postoji tradicija da prilikom učenja novog programskog jezika, prvi program koji napišete i pokrenete bude 'pozdrav Svima!' ('Hello World') program - sve što on radi je to da samo prikaže 'Pozdrav Svima!' kada ga pokrenete. Kako bi Simon Cozens (autor zadivljujuće "Beginning Perl" knjige) rekao, da je to "tradicionalna molitva programerskim bogovima da Vam pomognu da naučite jezik bolje."
Pokrenite izabrani editor, unesite sledeće program i sačuvajte ga kao `pozz.py`.
Ako koristite Komodo Edit, kliknite na `File - New - New File`, a zatim otkucajte sledeće:

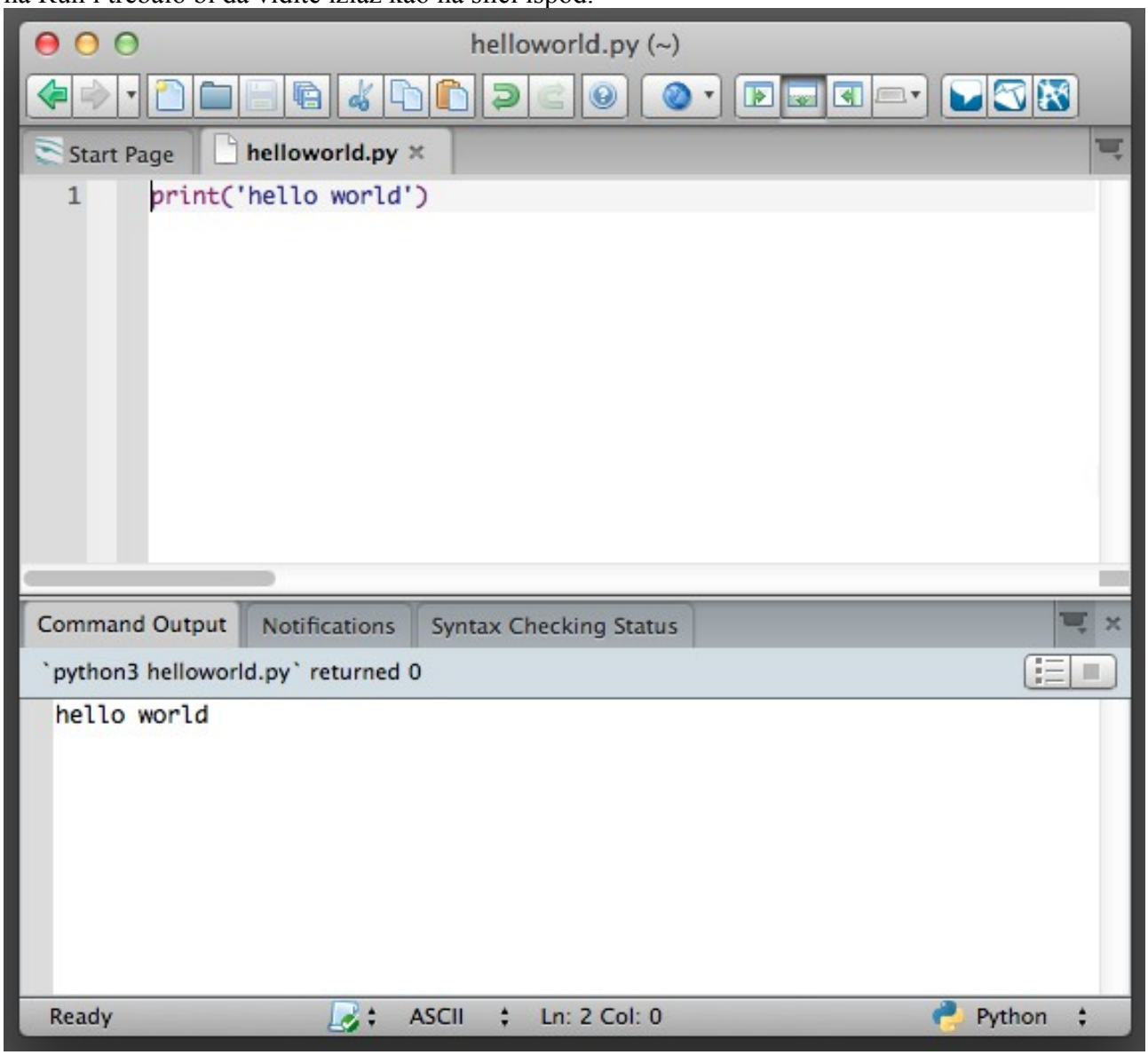
```
print('Pozdrav Svima!')
```

U Komodo Edit-u, kliknite `File - Save` da biste sačuvali tu datoteku.
Gde bi trebalo da sačuvate fajl? U bilo koji folder za koji znate njegovu lokaciju. Ako ne razumete šta to znači, kreirajte novu fasciklu i koriste njenu lokaciju da čuvate i pokrećete sve svoje Python programe:

- `C:\\py` na Windows-u
- `/tmp/py` na Linux-u
- `/tmp/py` na Mac OS X-u

Da biste kreirali folder, koristite `mkdir` komandu u terminalu, na primer, `mkdir /tmp/py` .
Važno
Uvek proverite da li ste datoteci dali ekstenziju `.py` , na primer, `foo.py` .

U Komodo Edit-u kliknite na `Tools - Run Command` , otkucajte `python3 pozz.py` i kliknite na Run i trebalo bi da vidite izlaz kao na slici ispod.



Snimak ekrana Komodo Edit-a sa Pozdrav Svetu programom

Bolji način je da, umesto da u Komodo Edit-u pokrećeto program, koriste terminal:

1. Otvorite terminal kao što je objašnjeno u poglavlju instalacije.
2. Promenite lokaciju tamo gde ste sačuvali datoteku, na primer, `cd /tmp/py`
3. Pokrenite program unošenjem komande `python3 pozz.py` .

Izlaz je prikazan ispod.

```
$ python3 pozz.py
Pozdrav Svima!
```

Ako imaš izlaz kao na slici gore, čestitam! - Uspešno ste pokrenuli svoj prvi Python program. Uspešno ste prešli najteži deo učenja programiranja, koji je, sam početak sa svojim prvim programom!

U slučaju da imate neku grešku, molimo vas da upišete gornji program *upravo onako kako je pokazano* iznad i ponovo pokrenite program. Imajte na umu da su slova u Python-u osetljiva na veličinu, odnosno `print` nije isto što i `Print` - obratite pažnju na malim p u prvom primeru i veliko P u drugom. Takođe, uverite se da nema tabova i razmaka pre prvog znaka u svakoj liniji - mi ćemo videti zašto je ovo važno kasnije.

Kako ovo funkcioniše

Svaki Python program se sastavlja od instrukcija (*statements*). U našem prvom programu, imamo samo jednu instrukciju. U toj instrukciji, mi pozivamo `print` komandu (*funkciju*) koja samo prikazuje tekst `'Pozdrav Svima!'`. Mi ćemo naučiti više o komandama i funkcijama detaljnije u kasnijim poglavljima - ono što za sada treba da shvate je da sve što postavite u zagradama će biti prikazano na ekranu. U ovom slučaju, mi smo postavili tekst `'Pozdrav Svima!'`.

5.3.1 Izvršavanje Python programa

Ovo važi samo za Linux i Unix korisnike, ali bi i korisnici Windows-a trebalo da znaju ovo.

Svaki put, kada želimo da pokrenemo Python program, moramo ga izričito pozvati komandom `python3 foo.py`, zašto ga ne bi mogli pokrenuti kao i svaki drugi program na našem računaru? To možemo postići pomoću nečega što se zove *hashbang* linija.

Dodajte liniju iz donjeg primera tako da bude *prva linija* u vašem programu:

```
#!/usr/bin/env python3
```

Dakle, vaš program bi sada trebalo da izgleda ovako:

```
#!/usr/bin/env python3
print ('Pozdrav Svima!')
```

Drugo, moramo dati izvršnu dozvolu programu koristeći `chmod` komandu,a zatim *pokrenite* izvorni program.

chmod komanda se ovde koristi za *promeni ponašanje* datoteke dajući joj *dozvolu* da se *izvršava* za sve korisnike sistema.

```
$ chmod a+x pozz.py
```

Sada možemo pokrenuti naš program direktno, jer je naš operativni sistem poziva `/usr/bin/env` što će pronaći naš Python 3 program i zato zna kako da pokrene naš izvorni fajl:

```
$ ./pozz.py
Pozdrav Svima!
```

Mi koristimo oznaku ./ da bi ukazali operativnom sistemu da se program nalazi u trenutnoj fascikli.

Da bi stvari bile još više zabavne, možete da preimenujete datoteku da bude samo `pozz` i pokrenuti je kao `./pozz` , i program će i dalje raditi, jer sistem zna da mora da se pokrene program pomoću interpretera čija lokacija je navedeno u prvom redu u izvornom fajlu.

Do sada smo bili u mogućnosti da pokrenemo svoj program dok znamo njegovu tačnu lokaciju. Šta ako želimo da možemo da pokrenemo program iz foldera? To možete da uradite tako što ćete sačuvati program u jednom od foldera navedenih u PATH promenljivih okruženja.

Kad god pokrenete bilo koji program, sistem traži taj program u svakom od direktorijuma navedenih u PATH promenljivoj okruženja, a zatim pokreće taj program. Možemo učiniti ovaj program dostupan svuda jednostavnim kopiranjem njegovog izvornog fajla u jedan od direktorijuma navedenih u PATH .

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/swaroop/bin
$ cp pozz.py /home/swaroop/bin/pozz
$ pozz
Pozdrav Svima!
```

Ovde smo prikazali PATH promenljivu korišćenjem echo komande sa prefiksom $ da ukažemo šelu da nam je potrebna vrednost ove "promenljive okruženja". Ovde vidimo da je `/home/swaroop/bin` jedan od direktorijuma iz promenljive PATH, gde je *swaroop* korisničko ime koje ja koristim na svom sistemu. Obično će postojati sličan direktorijum na vašem sistemu, samo sa korisničkim imenom koji Vi koristite.
Ako želite da dodate neki direktorijum po vašem izboru u PATH promenljivu - to može da se postigne kucajući `export PATH=$PATH:/home/swaroop/mydir` gde je '`/home/swaroop/mydir`' direktorijum koji želim da dodam u PATH promenljivu.
Ovaj metod je veoma koristan ako želite da pišete komande koje možete pokrenuti bilo kada, bilo gde. To je slično stvaranju sopstvenih komandi kao što je `cd` , ili bilo koja druga komanda koju koristite u terminalu.

5.4 Pronalaženje pomoći

Ako Vam je potrebna brza informacija o bilo kojoj funkciji ili komandi u Python-u, onda možete koristiti ugrađenu `help` funkciju. Ona je veoma korisna, posebno kada se koristi interaktivni prompt. Na primer, u interaktivnom promptu pokrenite `help(print)` - ova komanda prikazuje pomoć za print funkciju koja se koristi za prikazivanje stvari na ekranu.
Pažnja
Pritisnite taster q za izlazak iz pomoći.
Slično tome, možete dobiti informacije o gotovo svemu u Python-u. Kucajte `help()` da bi ste saznali više o korišćenju samog `help`-a!
U slučaju da morate da dobijete pomoć za operatore poput komande `return` , onda morate da stavite tu komandu unutar citata kao na primer `help('return')`, tako da se Python ne bi zabunio oko onoga šta od njega tražimo.

5.5 Rezime

Sada bi trebali da znate da napišete, sačuvate i pokrećete Python programe sa lakoćom.
Sada kada ste korisnik Python-a, hajde da naučite i neke naprednije koncepte Python-a.

6 Osnove

Samo prikazivanje 'Pozdrav Svima!' nije dovoljno, zar ne? Vi želite da uradite mnogo više od toga - vi želite da imate neke ulazne podatke, menjate ih na neki način i da dobijete nešto od svega toga. Mi možemo to postići u Python-u koristeći konstante i promenljive, a, pored toga, naučićemo i neke druge koncepte u ovom poglavlju.

6.1 Komentari

Komentari su svaki tekst koji pišemo sa desne strane # simbola i uglavnom se koriste kao napomena za one koji čitaju naš program.
Na primer:

```
print('Pozdrav Svima!')  # Primetite da je print funkcija
```

ili:

```
# Primetite da je print funkcija
print('Pozdrav Svima!')
```

Koristite onoliko korisnih komentara, koliko možete, u svom programu da:

- objasnite pretpostavke
- objasnite važne odluke
- objasnite važne detalje
- objasnite probleme koje pokušavate da rešite
- objasnite probleme koje ste pokušavali da izbegnete u svom programu, itd

Kod vam govori kako, komentari bi trebalo da vam kažu zašto.
Ovo je korisno za čitaoce vašeg programa, da bi oni lako mogli da razumeju šta program radi. Zapamtite, ta osoba (čitaoc programa) možete biti i Vi sami - ali posle šest meseci!

6.2 Doslovne konstante (Literal constants)

Primer doslovnih konstanti su brojevi kao na primer 5, 1.23 , ili stringovi (slova, reči) kao što su `'Ovo je string'` ili `"To je string!"` . One se zovu doslovne (*literalne*), jer bukvalno - koristite njihovu vrednost doslovno (onako kako ste odkucali). Broj 2 uvek predstavlja sebe i ništa drugo - on je *konstanta* jer njegova vrednost ne može biti promenjena. Dakle, sve ovo do sada što smo radili nosi naziv doslovna konstanta.

6.3 Brojevi

Brojevi se uglavnom dele na dve vrste - cele brojeve i decimale.
Jedan primer je: ceo broj (ili integer) 2 je samo vrednost koja predstavlja ceo broj.
Primeri brojeva sa pokretnim zarezom (ili decimali, ukratko *floats*) su 3.23 , i 52.3E-4 . E oznaka ukazuje na stepen broja 10. U ovom slučaju, 52.3E-4 znači $52.3 * 10^{-4}$.
Napomena za iskusne programere
Ne postoji poseban long tip brojeva. `int` tip brojeva može da bude ceo broj bilo koje veličine.

6.4 Stringovi (reči - strings)

String je *niz znakova*. Stringovi su u osnovi samo gomile reči.
Vi ćete koristiti stringove u skoro svakom Python programu koji budete pisali, pa obratite pažnju na deo koji sledi.

6.4.1 Između apostrofa

Možete odrediti string tako što ga stavite između apostrofa, na primer `'Apostrofi su oko mene'`. Svaki prazan prostor, odnosno razmak i uvlačenje (tabovanje) su prikazani onako kako su i napisani.

6.4.2 Između navodnika

Stringovi u navodnicima se ponašaju na isti način kao stringovi između apostrofa. Primer je `"Kako se zoves?"`

6.4.3 Između trostrukih apostrofa ili navodnika

Možete da odredite string koji sadrži više linija uz trostruke navodnike ili apostrofe - (""" ili '''). Čak možete slobodno koristiti apostrofe i navodnike u okviru trostrukih navodnika. Na primer:

```
'''Ovo je string iz vise linija. Ovo je prva linija.
Ovo je druga linija.
"Kako se zoves?" Pitao sam ga.
On rece "Bond 100's, Jovan Bond 100's."
'''
```

6.4.4 Stringovi se ne mogu menjati

To znači da kada ste kreirali neki string, ne možete da ga promenite. Iako ovo, na prvi pogled može izgledati kao loša stvar, to zaista i nije tako. Videćemo zašto to nije nikakvo ograničenje u različitim programima koje ćemo pisati kasnije.

Napomena za C/C++ programere

Ne postoji poseban `char` tip podataka u Python-u. Ne postoji realna potreba za tim i siguran sam da Vam neće nedostajati.

Napomena za Perl/PHP programere

Zapamtite da su stringovi koji su citirani i stringovi koji su pod navodnicima isti - oni se ne razlikuju, niti se ponašaju na bilo koji drugi način.

6.4.5 Metoda formatiranja

Ponekad želimo da izgradimo stringove od nekih drugih informacija. Na ovom mestu nam je `format()` koristan.

Sačuvajte sledeće linije kao fajl `formatiranje_stringa.py` (napomena: ukoliko koristite Linux možete u stringu koristiti i naše ćirilične/latinične fontove. To se dešava zato što je kod većine (ako ne i kod svih) distribucija podrazumevano kodiranje znakova UTF-8, dok na Windows operativnim sistemima to nije slučaj. U cilju što prostijeg objašnjavanja i UČENJA, u primerima, koji se nalaze u ovoj knjizi, su korišćeni "ćelavi" fontovi, tj slova su pisana bez karakterističnih kukica za naš jezik. Podešavanje komandne linije unutar Windows-a, kako bi ispravno prikazivala naše znakove, je izvan cilja ove knjige):

```
godine = 20
ime = 'Swaroop'
print('{0} je imao {1} godina, kada je napisao ovu knjigu.'.format(ime,
godine))
```

```
print('Zasto se {0} igra sa tim Python-om?'.format(ime))
```

Izlaz:

```
$ python3 formatiranje_stringa.py
Swaroop je imao 20 godina, kada je napisao ovu knjigu.
Zasto se Swaroop igra sa tim Python-om?
```

Kako to funkcioniše:
String može da koristi nekoliko specifikacija i samim tim, *format* metoda može biti pozvana da zameni te specifikacije sa odgovarajućim argumentima `format` metodoe.
Obratiti pažnju na prvu primenu gde smo koristili {0}, što odgovara promenljivoj `ime` koja je prvi argument u format metodu. Slično tome, druga specifikacija je {1} koja odgovara promenljivoj `godine` koja je drugi argument u format metodi. Imajte na umu da Python počinje brojanje od 0, što znači da je prva pozicija na indeksu 0, druga pozicija ima indeks 1, i tako dalje.
Primetimo da smo mogli postići istu stvar koristeći spajanje stringova: `ime + ' je ' + str(godine) + ' godina star'` , ali ova metoda je mnogo ružnija i podložnija greškama. Druga stvar, konverzija u string će se automatski izvršiti `format` metodom, umesto da je eksplicitno konvertujemo u string, što je potrebno u ovom slučaju. Treće, kada se koristi `format` metoda, možemo da promenimo poruku, bez potrebe da se bakćemo sa promenljivima koje se koriste i obrnuto.
Takođe imajte na umu da su brojevi u vitičastim zagradama opcionalni, tako da je isto ovo moglo da se napiše kao:

```
godine = 20
ime = 'Swaroop'

print('{} je imao {} godina, kada je napisao ovu knjigu.'.format(ime,
godine))
print('Zasto se {} igra sa tim Python-om?'.format(ime))
```

koji će dati isti izlaz kao i u prethodnom programu.
Ono što Python radi u `format` metodi je da zameni svaku vrednost argumenta na mestu specifikacije. Mogu postojati više detaljne specifikacije kao što su:

- decimalni broj (.) zaokružen na 3 cifre za decimalni broj (oznaka za decimalni broj - float) float '0.333'

```
>>> '{0:.3}'. format(1/3)
```

- okružuje sa donjim crtama (_) tekst koji će biti centriran (^) na dužinu od 11 karaktera '___Pozdrav___'

```
>>> '{0:_^11}'.format('Pozdrav')
```

- opis rečima 'Swaroop je napisao Zagrljaj Pitona'

```
>>> '{ime} je napisao {knjiga}'.format(ime = 'Swaroop', knjiga =
'Zagrljaj Pitona')
```

(Sve ove primere možete proveriti u interaktivnom prompt-u).

6.5 Promenljive

Korišćenje samo literalnih konstanti može uskoro postati dosadno - moramo na neki način obezbediti čuvanja informacije i manipulaciju istim tim informacijama. Ovo je mesto gde promenljive dolaze u centar pažnje. Promenljive su upravo ono što ime sugeriše - njihova vrednost može varirati, odnosno, možete sačuvati bilo šta koristeći promenljive. Promenljive su samo delovi memorije računara gde Vi skladištite neke informacije. Za razliku od literalnih konstanti, morate imati neki metod da pristupite ovim promenljivim i zato im dajemo neka imena.

6.6 Imenovanje promenljive (identifikacija)

Promenljive su primeri identifikatora. *Identifikatori* su imena, koja su data, da bi identifikovali *nešto*. Postoje neka pravila koje morate da sledite pri imenovanju identifikatora:

- Prva oznaka u imenu identifikatora mora da bude slovo azbuke (velike ili male ASCII ili Unicode oznake) ili donja crta ('_').
- Ostale oznake u imenu identifikatora mogu da se sastoje od slova (velike ili male ASCII ili Unicode oznake), donje crte ('_') ili cifre (0-9).
- Imena identifikatora su osetljiva na veličinu slova. Na primer, `mojeime` i `mojeIme` **nisu iste** promenljive. Obratite pažnju na malo i u prvoj oznaci i veliko slovo I u drugoj.
- Primeri *pravilnih* imena identifikatora su `i`, `__moje_ime`, `ime_23`. Primeri "*nevažećih*" imena identifikatora su `2stvari`,`ovo je napisano sa razmacima`, `moje-ime`,`>a1b2_c3` i `"ovo_je_pod_navodnicima"`.

6.7 Tipovi podataka

Promenljive mogu imati vrednosti različitih tipova koje zovemo **tipovi podataka** (data types). Osnovni tipovi su brojevi i stringovi, o kojima smo već govorili. U kasnijim poglavljima, videćemo kako da kreirate sopstvene tipove podataka koristeći klase (class).

6.8 Objekat

Zapamtite, Python se odnosi prema svemu što smo uneli u program kao *objekatu*. To je rečeno u generičkom smislu. Umesto da kažemo "*nešto*", mi kažemo "*objekat*".

Napomena za Objektno Orijentisane programere

Python je strogo objektno orijentisan u smislu da je sve objekat, uključujući brojeve, stringove i funkcije.

Sada ćemo videti kako da koristite promenljive zajedno sa literalnim konstantama. Sačuvajte sledeći primer i pokrenite program.

6.9 Kako pisati Python programe

Ubuduće, standardna procedura da sačuvate i pokrenete Python program je sledeći:

1. Otvorite svoj izabrani editor, npr Komodo Edit.
2. Prepišite kod programa datog u primeru.
3. Sačuvajte ga kao datoteku sa imenom koje je pomenuto.

4. Pokrenite interpreter sa komandnim `python3 ime_programa.py` da pokrenete program.

6.10 Primer: Korišćenje promenljivih i literalnih konstanti

Ime programa: `var.py`

```
i = 5
print(i)
i = i + 1
print (i)
s = '''Ovo je string u vise linija.
Ovo je druga linija.'''
print(s)
```

Izlaz:

```
$ python3 var.py
5
6
Ovo je string u vise linija.
Ovo je druga linija.
```

Kako to funkcioniše:
Evo kako ovaj program radi. Prvo smo dodelili literalnu konstantu vrednosti 5 promenljivoj i pomoću operatora dodele (=). Ova linija se zove naredba (statement), jer se u njoj kaže da nešto treba da se uradi, u ovom slučaju, dodeliti promenljivoj, koja se zove i, vrednost 5. Dalje, mi smo prikazali vrednost promenljive i korišćenjem `print` funkcije koja, što nije iznenađujuće, samo prikazuje vrednost promenljive na ekranu.
Zatim smo dodali 1 na vrednosti koja se nalazi u `i` i sačuvali ga nazad u promenljivu `i`. Zatim smo ga prikazali i očekivano, dobijamo vrednost 6 .
Slično tome, mi smo dodelili literalnu konstantu koja je, ništa drugo nego string, promenljivoj s , a zatim je prikazali na ekranu.
Napomena za programere na statičnim jezicima
Promenljive se koriste samo da bi im dodeli vrednost. Nije potrebno da se vrši definicija tipa podataka, niti ikakva deklaracija.

6.10.1 Logička i fizička linija

Fizička linija je ono što *vidite* kada pišete program. Logička linija je ono što *Python vidi* kao jednu naredbu. Python implicitno pretpostavlja da svaka *fizička linija* odgovara *logičnoj liniji*.
Primer logičke linije je izjava poput `print('Pozdrav Svima!')`. Ako je to prikazano na jednoj liniji (kako ga vidite u editoru), onda to takođe odgovara i fizičkoj liniji.
Implicitno, Python ohrabruje da koristimo jednu izjavu po liniji,što čini naš kod lakše čitljivim.
Ako želite da navedete više od jednog logičkog reda na jednoj fizičkoj liniji, onda morate da izričito "kažete" to Python-u, koristeći tačku-zarez (;) koji pokazuje na kraj logičke linije/izjave. Na primer:

```
i = 5
print(i)
```

je efektivno isto kao i

```
i = 5;
```

```
print(i);
```

a isto se može napisati kao

```
i = 5; print(i);
```

ili čak

```
i = 5; print(i)
```

Međutim, **ja preporučujem** da se držimo **pisanja maksimalno jedne logičke linije na svaku pojedinačnu fizičku liniju**. Ideja je da nikada ne treba da koristite tačku-zarez. U stvari, *nikada* nisam koristio, ili čak video tačku-zarez u nekom Python programu.
Postoji jedna situacija u kojoj je ovaj koncept stvarno koristan: ako imate dugačku liniju koda, možete je razbiti na više fizičkih linija koristeći obrnutu kosu crtu (\). Ovo se naziva **eksplicitno spajanje linija**:

```
s = 'Ovo je string. \
Ovo je nastavak string-a.'
print(s)
```

Ovo daje izlaz:

```
Ovo je string. Ovo je nastavak string-a.
```

Slično tome,

```
print\
(i)
```

je isto kao i

```
print(i)
```

Ponekad, postoji implicitna pretpostavka gde Vi ne morate da koristite obrnutu kosu crtu. To je slučaj kada logička linija ima početnu zagradu, početnu uglastu zagradu ili početnu vitičastu zagradu, ali ne i krajnju. To se zove **implicitno spajanje linija**. Možete to videti u akciji kada budemo pisali programe koristeći liste u kasnijim poglavljima.

6.10.2 Uvlačenje

Razmak je važan u Python-u. Zapravo, **razmaci na početku linija su važni**. Ovo se zove **indentacija** (uvlačenje). Razmaci na početku (razmaci i tabovi) koji se nalaze na početku logičke linije se koriste da se odredi nivo indentacije logičke linije, što se, kako se ispostavilo, koristi da se odredi grupisanje (blokova) naredbi.
To znači da komande koje idu zajedno, **moraju** da imaju isti nivo uvlačenja. Svaki takav skup komandi se naziva **blok**. Videćemo primere zašto su blokovi važni u kasnijim poglavljima.
Jedna stvar koju treba zapamtiti je da pogrešno uvlačenje dovodi do grešaka. Na primer:

```
i = 5
 print('Vrednost je ' , i) # Greska! Primetite jedan razmak na pocetku
# linije
print('Ponavljam, vrednost je ' , i)
```

Kada pokrenete ovo, dobićete sledeću grešku:

```
File "whitespace.py", line 4
 print('Value is ', i) # Greska! Primetite jedan razmak na pocetku
 ^

IndentationError: unexpected indent
```

Obratite pažnju da postoji jedan razmak na početku drugog reda. Greška na koju ukazuje Python nam govori da je sintaksa programa pogrešna, odnosno program nije ispravno napisan. Šta to znači za vas? Znači da *ne možete započeti nove blokove naredbi proizvoljno* (osim za podrazumevani glavni blok koji ste koristili sve do sada, naravno). Slučajevi u kojima možete da koristite nove blokove biće detaljno opisani u kasnijim poglavljima, kao što je kontrola protoka.

Kako da uvučete

Koristite samo razmake za uvlačenje, ako koristite tabove, budite sigurn da je podešen na veličinu od 4 razmaka. Dobri editori kao što je Komodo Edit će automatski da uvlače potrebne delove programa, umesto vas. Uverite se da koristite isti broj mesta za uvlačenje (4 razmaka prvi blok, 8 drugi itd...), u suprotnom, vaš program će pokazati greške.

Napomena za programere statičkih jezika

Python će uvek koristiti uvlačenje za blokove i nikada neće koristiti vitičastu zagradu. Pokreni `from __future__ import braces` da saznate više.

6.11 Rezime

Sada kada smo prošli kroz mnoge dosadne detalje, možemo da pređemo na mnogo interesantnije stvari poput naredbi kontrole tokova (control flow statements). Budite sigurni da ste dobro naučili ono što ste pročitali u ovom poglavlju.

7 Operatori i Ekspresije

Većinu naredbi (logičkih linija) koje budete pisali će sadržati *ekspresije* (matematičke izraze). Jednostavan primer izraza je 2 + 3 . Izraz se može razložiti na operatore i operande.
Operator je funkcionalnost koja radi nešto, i može biti predstavljen simbolima, kao što je + ili posebnim rečima (oznakama). Operatorima su potrebni neki podatci da bi izvršili neku radnju nad njima i takvi podaci se zovu *operandi*. U ovom slučaju, 2 i 3 su operandi.

7.1 Operatori

Ukratko ćemo da pojasnimo operatere i njihovu upotrebu:
Imajte na umu da možete da proverite izraze date u primerima koristeći interaktivni prompt. Na primer, da testirate izraz 2 + 3 , koristite interaktivni Python prompt:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

+ **(plus)** Dodaje dva objekta
`3 + 5` daje `8` . `'a' + 'b'` daje `'ab'`.
- (minus) Oduzima jedan broj od drugog, ako prvi operand nije naznačen pretpostavlja se da je nula.
`-5.2` daje negativan broj i `50 - 24` daje `26` .
* **(množenje)** množenje dva broja ili vraća string ponovljen dat broj puta.
`2 * 3` daje `6` . `'la' * 3` daje `'lalala'` .
** **(stepen)** Vraća x stepenovan sa y
`3 ** 4` daje `81` (tj. `3 * 3 * 3 * 3`)
/ **(deljenje)** Podeli x sa y
`4 / 3` daje `1.3333333333333333` .
// **(celobrojna vrednost rezultata podele)** Vraća ceo broj od rezultata deljenja
`4 // 3` daje `1` .
% **(modulo)** Daje ostatak podele dva broja
`8 % 3` daje `2` . `-25.5 % 2.25` daje `1.5` .
<< **(pomeranje ulevo)** Pomera bitove od datog broja ulevo za broj preciziranih bitova. (Svaki broj je zastupljen u memoriji računara kao skup bitova ili binarna cifra, odnosno skup 0 i 1)
`2 << 2` daje `8` . 2 predstavlja 10 u bitovima (binarni oblik).
Pomeranje ulevo od 2 bita daje 1000 što predstavlja decimalni broj 8 .
>> **(pomeranje udesno)** Pomera bitove datog broja udesno za broj bitova navedenih u drugom broju.
`11 >> 1` daje `5` .
11 je predstavljen u bitovima sa 1011 što kada pomeraimo u desno za 1 bit daje 101 koji je decimalni broj 5.
& (logičko I nad bitovima) logičko I nad brojevima koji su predstavljeni bitovima (slično kao i pomeranje bitova iz gornjeg primera, samo što se ovde vrši logička operacija nad tim bitovima)
`5 & 3` daje `1` .
| **(logičko ILI nad bitovima)** logičko ILI nad bitovima brojeva
`5 | 3` daje `7`
^ **(logičko KSILI nad bitovima)** logičko KSILI nad bitovima brojeva

`5 ^ 3` daje `6`
~ (logička inverzija bitova) Logička inverzija x je - (x +1)
`~5` daje `-6` .
< (manje od) Proverava da li je x manje od y. Svi operatori koji upoređuju vrednosti vraćaju vrednosti `True` ili `False` . Obratite pažnju na velika slova u imenima vrednosti.
`5 < 3` daje `False` a `3 < 5` daje `True`.
Poređenja mogu biti produžena proizvoljno: `3 < 5 < 7` će dati `True` .
> (veće od) Poredi da li je x veći od y
`5 > 3` vraća `True` . Ako su oba operanda brojevi, oni su prvo konvertuju u zajednički tip. U suprotnom, uvek vraća `False`.
<= (manje ili jednako) Poredi da li je x manje ili jednako y
`x = 3; y = 6; x <= y` vraća `True`.
>= (veće ili jednako) Poredi da li je x veći ili jednako y
`x = 4; y = 3; x >= y` vraća `True` .
== (da li je jednako) Poredi da li su objekti jednaki
`x = 2; y = 2; x == y` vraća `True` .
`x = 'str'; y = 'stR'; x == y` vraća `False` .
`x = 'str'; y = 'str'; x == y` vraća `True` .
!= (nije jednako) Poredi da li objekti nisu jednaki
`x = 2; y = 3; x != y` vraća `True` .
not (bulovo NE) Ako je x `True` , on vraća `False` . Ako je x `False` , vraća `True` .
`x = True; not x` vraća `False` .
and (Bulovo I) `x and y` vraća `False` ako je x `False` , inače vraća vrednost y
`x = False; y = True; x and y` vraća `False` pošto je x `False.` U ovom slučaju, Python neće uopšte proveravati vrednost y, jer zna da izraz sa leve strane 'and' ima vrednost `False` što automatski znači da će ceo izraz da bude `False` nezavisno od drugih vrednosti. To se zove evaluacija u kratkom spoju.
or (Bulovo ILI) Ako je x `True` , vraća `True`, inače vraća vrednost y
`x = True; y = False; x or y` vraća `True` . Evaluacija kratkog spoja važi i u slučaju operacije `or`.

7.1.1 Prečice za matematike operacije i dodavanja vrednosti

Uobičajeno je da se traži neka matematička operacija nad promenljivom, a zatim da se "vrati" vrednost rezultata te operacije nazad u promenljivu, pa postoji prečica za takve izraze:
Možete napisati:

```
a = 2
a = a * 3
```

kao:

```
a = 2
a *= 3
```

Primetimo da "`var = var operacija izraz`" postaje "`var operacija = izraz`".

7.2 Koje evaluacije imaju proritet?

Ako ste imali izraz kao što je `2 + 3 * 4` , da li se vrši prvo sabiranje ili množenje? Naša

matematika iz viših razreda osnovne škole, nam govori da treba da se prvo uradi množenje. To znači da operator množenja ima veći prioritet od operatora sabiranja.
Sledeća tabela predstavlja prvenstvo operacija u Python-u, od najnižeg prioriteta (najkasnije se izvršava) do najvišeg prioriteta (najranije se izvršava). To znači da će u datom izrazu, Python prvo izvršiti operatore i ekspresije koji se nalaze niže u tabeli, pre nego što će izvršiti one što su navedeni iznad njih u tabeli.
Sledeća tabela preuzeta sa Python reference manual, ovde je priložena radi kompletnosti, jer je ipak daleko pametnije koristiti zagrade za grupisanje operatora i operanda na željeni način kako bi se eksplicitno naveo prednost izvršavanja. Tako grupisani čine naš program lakše čitljiv. Pogledajte Promena redosleda evaluacije za detalje.
lambda Lambda izraz
or Bulova ILI
and Bulova I
not x Bulova NE
in, not in Test pripadnosti
is, is not Test identiteta
<, <=, >, >=, !=, == Poređenja
| Logičko (nad bitovima) ILI
^ Logičko (nad bitovima) KSOR
& Logičko I nad bitovima
<<, >> Pomeranja
+, - Sabiranje i oduzimanje
***, /, //, %** Množenje, deljenje, celobrojno deljenje i Ostatak
+x, -x Pozitivan, negativan
~x Logičko NE nad bitovima
****** Stepenovanje
x.atribut Referenca atributa
x[index] subskripcija
x[index1:index2] Sečenje liste
f(argument ...) Poziv funkcije
(ekspresija, ...) Prikazivanje ili dodela vrednosti tuple-ima (prikazivanje vrednosti, ili dodela vrednosti više promenljivih u isto vreme)
[ekspresija, ...] prikaz liste
{kljuc:datum, ...} prikaz rečnika
Operatori na koje nismo još naišli će biti objašnjeni u narednim poglavljima.
Operateri koji imaju *isto prvenstvo* su navedeni u istom redu u gornjoj tabeli. Na primer, + i - imaju istu prednost prvenstva izvršavanja.

7.3 Promena redosleda izvršavanja evaluacije

Da bi izraze učinili lakše čitljivim, možemo koristiti zagrade. Na primer, `2 + (3 * 4)` je definitivno lakše razumeti nego `2 + 3 * 4` jer ovako napisano, zahteva znanje redosleda izvršavanja operatora. Kao i u svemu drugom, zagrade treba umereno koristiti (ne preterujte) i ne bi trebalo da ih bude suvišnih, kao u `(2 + (3 * 4))` .
Postoji dodatna prednost korišćenja zagrada - one nam pomažu da se promeni redosled izvršavanja evaluacije. Na primer, ako želite da dodavanje bude izvršeno pre množanja u našem primeru, onda možete napisati ovako `(2 + 3) * 4` .

7.4 Asocijativnost (dodela vrednosti)

Operatori se obično asociraju (dodeljuje im se vrednost) sa leva na desno. To znači da su operatori sa istim prvenstvom izvršavanja evaluiraju (izvršavaju se) sa leve na desnu stranu. Na primer, `2 + 3 + 4` se izvršava kao `(2 + 3) + 4` . Neki operateri poput dodele vrednosti (=), imaju s desna na levo asocijativnost tj `a = b = c` tretira se kao `a = (b = c)` .

7.5 Izrazi (ekspresije)

Primer (sačuvati kao `ekspresije.py`):

```
duzina = 5
sirina = 2

povrsina = duzina * sirina
print('Povrsina je', povrsina)
print('Obim je', 2 * (duzina + sirina))
```

Izlaz:

```
$ python3 ekspresije.py
Povrsina je 10
Obim je 14
```

Kako to funkcioniše:
Dužina i širina pravougaonika se čuvaju u promenljivim istog imena. Mi ih koristimo za izračunavanje površine i obima pravougaonika uz pomoć izraza. Čuvamo rezultat izraza `duzina * sirina` u promenljivoj `povrsina` , a zatim je prikažemo pomoću `print` funkcije. U drugom slučaju, mi smo direktno koristiti vrednost izraza `2 * (duzina + sirina)` u funkciji za prikaz (print).
Takođe, primetite kako Python 'lepo štampa' izlaz. Iako nismo naveli razmak između `'Povrsina je'` i promenljive `povrsina` , Python je to uradio za nas, tako da smo dobili čist i lep izlaz, a i program je mnogo čitljiviji na ovaj način (jer ne moramo da brinemo o razmaku u stringovima koje koristimo, prilikom prikaza). Ovo je primer kako Python može da učini život lakši za programera.

7.6 Rezime

Videli smo kako da koristimo operatore, operanade i izraze - a to su osnovni gradivni blokovi bilo kojeg programa. U sledećem poglavlju mi ćemo videti kako da iskoristimo to u našim programima koristeći naredbe (statements).

8 Kontrola protoka

U programima koje smo videli do sada, uvek je postojao niz naredbi, koje se verno izvršavaju od strane Python-a u tačnom poretku od vrha naniže. Šta ako ste želeli da promenite tok izvršavanja programa? Na primer, želite da program donosi neke odluke i radi različite stvari u zavisnosti od različitih situacija, kao što je prikaz reči 'Dobro jutro' ili 'Dobro veče", u zavisnosti od doba dana?
Kao što ste možda pogodili, to se postiže korišćenjem komandi kontrole protoka. Postoje tri komande za kontrolu tokova u Python-u - `if` , `for` i `while` .

8.1 If komanda

`if` komanda se koristi da proverite uslov: *ako* je uslov istina, mi pokrećemo blok naredbi (nazvan *if-blok*), *inače* ćemo obraditi sledeći blok naredbi (nazvan *else-blok*). Druga klauzula je opcionalna (znači da else-blok *nije obavezan*).
Primer (sačuvati kao `if.py`):

```
broj = 23
pogadjanje = int(input('Pogodi broj: '))

if pogadjanje == broj:
    print('Cestitamo, pogodio si ga.')  # Novi blok pocinje ovde
    print('(Ali nisi osvojio nikakvu nagradu!)')  # Novi blok se
# zavrsava ovde
elif pogadjanje < broj:
    print('Ne, on je malko veci nego taj.')  # Jos jedan blok
    # Mozete raditi sta vam je volja u ovom bloku
else:
    print('Ne, on je malko manji nego taj')
    # Da bi se izvrsio ovaj blok moralo je biti pogadjanje > broj.
print('''Gotovo :)''') # Ovaj red ce se uvek izvrsavati posle if
# provere, jer je u "glavnom" bloku
```

Izlaz:

```
$ python3 if.py
Pogodi broj: 50
Ne, on je malko manji nego taj
Gotovo :)

$ python3 if.py
Pogodi broj: 22
Ne, on je malko veci nego taj.
Gotovo :)

$ python3 if.py
Pogodi broj: 23
Cestitamo, pogodio si ga.
(Ali nisi osvojio nikakvu nagradu!)
Gotovo :)
```

Kako to funkcioniše:
U ovom programu, uzimamo nagađanja od strane korisnika i proverimo da li je to broj koji smo mu zadali. Mi postavljamo vrednost promenljive `broj` na bilo koji ceo broj koji mi želimo, recimo 23. Onda pitamo za pretpostavku korisnika korišćenjem `input()` funkcije. Funkcije su samo višekratne komade programa (tj. delovi programa koji možemo koristiti na više različitih mesta u samom tom programu). Vi ćete pročitati više o njima u sledećem poglavlju.
Mi postavljamo string (rečenicu) u `input` funkciju, koja je prikazuje na ekranu i čeka unos od korisnika. Kada smo ukucali nešto i pritisnite enter taster, `input()` vraća programu, ono što smo ukucali, kao string. Mi onda konvertujemo taj string u ceo broj pomoću `int` a zatim ga sačuvamo u promenljivoj `pogadjanje` . Zapravo, `int` je klasa, ali sve što treba da znate je da možete da ga koristite za konvertovanje stringa u ceo broj (pod pretpostavkom da je string sadrži važeći ceo broj kao tekst, npr '6', '15' itd...).
Dalje, mi smo uporedili pogodak od korisnika sa brojem koji smo izabrali. Ako su jednaki, mi prikazujemo poruku o uspešnom pogotku. Primetite da koristimo nivoe uvlačenja da kažemo Python-u koje izjave pripadaju kojem bloku. To je razlog zašto je toliko važno uvlačenje (razmak) u Python-u. Nadam se da ste zalepili za pravilo "doslednog uvlačenjs". Jeste li?
Obratite pažnju da komanda `if` sadrži dvotačku na kraju - mi time kažemo Python-u da sledi neki blok naredbi.
Zatim smo proverili da li je pretpostavka manja od izabranog broja, i ako je tako, mi obavestimo korisnika da mora da pogodi broj koji je malo veći od toga. Ono što smo ovde koristi je `elif` klauzula koja zapravo kombinuje dve vezane `if else-if else` naredbe u jednu kombinovanu `if-elif-else` .To nam čini program lakšim i smanjuje količinu nepotrebnog uvlačenja.
`elif` i `else` naredbe moraju imati i dvotačku na kraju logičke linije praćenu odgovarajućim blokom naredbi (uz pravilno uvlačenje, naravno).
Možete imati još `if` komandi unutar if-bloka, sačinjenu od `if` naredbe, i tako dalje - to se zove ugnježđenje `if` komandi.
Zapamtite da `elif` i `else` delovi koda su opcioni. Minimalna validna if naredba:

```
if True :
    print ( 'Da, tacno je.' )
```

Nakon što je Python završio izvršavanje `if` komande zajedno sa pridruženom `elif` i `else` klauzulama, on prelazi na sledeću komandu u bloku koji je sadržao `if` naredbu. U ovom slučaju, to je glavni blok (gde je izvođenje programa i počelo), i sledeća naredba je `print('''Gotovo :)''')` . Nakon toga, Python vidi kraj programa i jednostavno ga završi.
Iako je ovo vrlo jednostavan program, ja sam istakao mnogo stvari koje Vi treba da primetite. Sve su to prilično jednostavne stvari (iznenađujuće jednostavne za neke od vas koji dolaze iz C/C++ sveta). Moraćete da budete pažljivi oko svih ovih stvari u početku, ali posle malo prakse Vaš rad će postati prijatan sa njima, i sve će postati prilično 'prirodno' za vas.
Napomena za C/C++ programere
Nema `switch` komande u Python-u. Možete koristiti `if..elif..else` naredbe da uradi istu stvar (i, u nekim slučajevima, koristite rečnik (dictionary) da to uradite brzo).

8.2 While naredba

`while` naredba vam omogućava da se konstantno izvršava neki blok komandi, dokle god uslov (zbog koga se while i poziva) je `True` (tačan). `while` naredba je primer onoga što se zove *ponavljajuća* komanda (looping statement). `while` naredba može imati opcionalan `else` deo.
Primer (sačuvati kao `while.py`):

```
broj = 23
radim = True

while radim:
    pogodi = int(input('Pogodi broj: '))

    if pogodi == broj:
        print('Cestitam, pogodio si.')
        radim = False  # Ovo ce prekinuti while petlju i preci na neki
# red posle nje
    elif pogodi < broj:
        print('Ne, broj je malo veci od tvog.')
    else:
        print('Ne, broj je malo manji od tvog.')
else:
    print('While petlja je gotova.')
    # Ovde mozete da radite sta Vam je volja

print('Kraj.')
```

Izlaz:

```
$ python3 while.py
Pogodi broj: 50
Ne, broj je malo manji od tvog.
Pogodi broj: 22
Ne, broj je malo veci od tvog.
Pogodi broj: 23
Cestitam, pogodio si.
While petlja je gotova.
Kraj.
```

Kako to funkcioniše:

U ovom programu, još uvek se igramo igru pogađanja broja, ali prednost je u tome što korisnik može da nastavi pogađanje sve dok ne pogodi - nema potrebe da stalno pokreće program za svako pogađanje, kao što je bio program u prethodnom odeljku . Ovaj primer najbolje demonstrira korišćenje `while` naredbe.

Mi smo pomerili `input` i `if` komande unutar `while` petlje i podesili promenljivu `radim` na `True` pre while petlje. Prvo, mi smo proverili da li promenljiva `radim` je zaista `True` , a zatim nastavili da izvršavamo odgovarajući *while-blok*. Nakon izvršavanja celog tog bloka, uslov se ponovo proverava, a to je u ovom slučaju `radim` promenljiva. Ako je uslov tačan, ponovo će se izvršiti while-blok, u suprotnom ćemo nastaviti da izvršavamo opcioni else-blok, a zatim prelazimo na sledeću komandu.

`else`-blok se izvršava kada uslov za `while` petlju postane `False` - to može biti čak i prvi put kada se stanje uslova proverava. Ako postoji `else` klauzula za `while`-petlju, ona će se uvek izvršavati osim ako iz while petlje ne "iskočimo" sa `break` komandom.

`True` i `False` se nazivaju Bulove vrednosti (po Bulovoj algebri) i možete ih smatrati ekvivalentima vrednosti 1 i 0 (1-True, 0-False).

Napomena za C/C++ programere
Zapamtite da možete imati `else` klauzulu za `while` petlje.

8.3 for petlja

`for..in` komanda je još jedna komanda za petlju koja *prolazi* kroz niz predmeta, odnosno prolaze kroz svaku stavku u nizu. Mi ćemo videti nešto više o sekvencama (neki predmeti u nekom nizu) detaljno u kasnijim poglavljima. Ono što treba da znate za sada je to da je sekvenca samo kolekcija (nekih) stavki u (nekom) određenom rasporedu.
Primer (sačuvati kao `for.py`):

```
for i in range(1, 5):
    print(i)
else:
    print('Petlja for je gotova.')
```

Izlaz:

```
$ python3 for.py
1
2
3
4
Petlja for je gotova.
```

Kako to funkcioniše:
U ovom programu, mi smo prikazali niz (*sekvencu*) brojeva. Mi generišemo niz brojeva pomoću ugrađene `range` funkcije.
Ono što mi ovde radimo je to da zadajemo dva broja u `range` funkciju, koja nam vraća niz brojeva koji počinju od prvog broja, ali do drugog broja (znači da drugi broj nije uključen u niz). Na primer, `range(1,5)` daje nam sekvencu `[1, 2, 3, 4]` . Podrazumevano, `range` broji (kreira sekvencu) sa korakom vrednosti od 1. Ako mi postavimo treći broj u `range` funkciju, onda on postaje korak brojanja. Na primer, `range(1,5,2)` daje nam `[1,3]` (tj. 1, 1+2). Zapamtite da opseg koji želimoda dobijemo, se proteže *DO* drugog broja, odnosno da **ne sadrži** taj drugi broj.
Imajte na umu da `range()` generiše niz brojeva, ali će on generisati samo jedan broj u jednom trenutku, kada od njega for-petlja to bude zahtevala za sledeći prolaz kroz niz komandi. Ako želite da odmah vidite punu sekvencu brojeva, koristite `list(range())`. Liste su objašnjene u [poglavlju o strukturi podataka].
`for` petlja zatim ponavlja svoje komande u ovom opsegu - za `for i in range(1,5)` ekvivalentno je `for i in [1, 2, 3, 4]` , što je dodeljivanje svakog broja (ili objekta) u nizu, promenljivoj i, i to jednog u jednom trenutku, a zatim izvršava blok naredbi za svaku vrednost `i` . U ovom slučaju, mi u bloku izjava imamo komandu koja samo ispiše vrednost promenljive.
Zapamtite da `else` deo nije obavezan. Ali kada smo ga definisali, uvek se izvršava jednom pošto se `for`-petlja završi, osim ako je nismo prekinuli sa `break` naredbom.
Zapamtite da `for..in` petlje rade za svaku sekvencu. Ovde imamo listu brojeva koje generiše ugrađena `range` funkcija, ali generalno možemo koristiti bilo koji niz bilo koje vrste objekata! Mi ćemo istražiti ovu ideju detaljno u kasnijim poglavljima.
Napomena za C/C++/Java/C# programere
Python `for` petlja je radikalno drugačija od C/C++ `for` petlje. C# programeri će primetiti da `for`-petlja u Python-u je slična `foreach` petlji u C#. Java programeri će primetiti da je slična `for`

`(int i : IntArray)` u Javi 1.5.
U C/C++, ako želite da pišete `for (int i = 0; i < 5; i++)` , to bi smo u Python-u napisati kao `for i in range(0,5)` . Kao što možete da vidite, `for`-petlja je jednostavnija, više izražajna i manje sklona greškama u Python-u.

8.4 break komanda

`break` komanda se koristi da bi se *prekinulo* izvršavanje neke petlje, odnosno komanda momentalno zaustavi izvršenje komande petlje, čak i ako uslov za petlju nije postao `False` ili prolazak kroz sekvencu stvari nije potpuno završen.
Važna napomena je da ako se *prekine* `for` ili `while` petlja sa komandom `break`, ako postoji, za odgovarajuću petlju, `else`, `else`-blok se **NEĆE** izvršiti.
Primer (sačuvati kao `break.py`):

```
while True:
    s = input('Ukucaj bilo sta: ')
    if s == 'izlaz':
        break
    print('Duzina stringa je', len(s))
print('Kraj')
```

Izlaz:

```
$ python3 break.py
Ukucaj bilo sta: Programiranje je zabavno
Duzina stringa je 24
Ukucaj bilo sta: Kada se zavrsi posao
Duzina stringa je 20
Ukucaj bilo sta: ako zelite da vas posao bude zabavan
Duzina stringa je 36
Ukucaj bilo sta:          koristi Python!
Duzina stringa je 24
Ukucaj bilo sta: izlaz
Kraj
```

Kako to funkcioniše:
U ovom programu, mi konstantno tražimo od korisnika da ukuca nešto, a zatim prikazujemo dužinu ukucanog stringa. Napravili smo i poseban uslov da bi se zaustavio program na taj način što korisnik jednostavno ukuca `'izlaz'` . Mi zaustavljamo program sa `break` komandom koja će program "*izbaciti*" iz petlje, i zatim dolazimo do kraja programa.
Dužina ulazne niske može se saznati pomoću ugrađene `len` funkcije.
Zapamtite da `break` naredba može da se koristi isto tako i sa `for`-petljom.

8.4.1 Swaroop-ova pesma o Python-u

Ulaz koji sam koristio u ovom primeru je mini pesma koju sam napisao i nazvao *Swaroop's Poetic Python* (ovde je napisana u originalu):
"Programming is fun
When the work is done
if you wanna make your work also fun:

use Python!"

8.5 continue komanda

`continue` komanda se koristi da kažemo Python-u da preskoči ostatak naredbi u tekućem bloku unutar petlje i da *nastavi* sa sledećim izvršavanjem te iste petlje.
Primer (sačuvati kao `continue.py`):

```
while True:
    s = input('Ukucaj nesto: ')
    if s == 'izlaz':
        break
    if len(s) < 3:
        print('Previse je kratko.')
        continue
    print('Input je zadovoljavajuce duzine.')
    # Mozete zadavati druge komande za neki rad ovde
```

Izlaz:

```
$ python3 continue.py
Ukucaj nesto: a
Previse je kratko.
Ukucaj nesto: 12
Previse je kratko.
Ukucaj nesto: abc
Input je zadovoljavajuce duzine.
Ukucaj nesto: izlaz
```

Kako to funkcioniše:
U ovom programu, tražimo unos od korisnika, ali, mi želimo da obradimo ulazni string samo ako je najmanje 3 karaktera dugačak. Dakle, mi koristimo ugrađenu `len` funkciju da bi dobili dužinu stringa i ako je dužina manja od 3, preskočimo ostatak komandi unutar bloka pomoću `continue` naredbe. U suprotnom, ostale naredbe u petlji se izvršavaju, radeći bilo kakvu vrstu obrade koju mi želimo da uradimo.
Imajte na umu da `continue` komanda isto može da radi i unutar `for`-petlje.

8.6 Rezime

Videli smo kako da koristimo tri izjave kontrole programa - `if` , `while` i `for` zajedno sa svojim pridruženom `break` i `continue` naredbama. Ovo su neke od najčešće korišćenih komandi u Python-u i samim tim, upoznajte se dobro sa njima, jer su od suštinskog značaja za pisanje programa.
U sledećem poglavlju ćemo videti kako da kreiramo i koristimo funkcije.

9 Funkcije

Funkcije su delovi programa koji se mogu koristiti na više različitih mesta unutar samog tog programa. One nam omogućavaju da damo ime bloku komandi, što nam dalje, omogućava da pokrenemo taj blok komandi koristeći to ime koje smo joj dodelili, bilo gde u svom programu i bilo koliki broj puta. Ovo je poznato kao *pozivanje* funkcije. Mi smo već koristili mnoge ugrađene funkcije kao što su `len` i `range` .
Koncept funkcija je verovatno *najvažniji* kamen temeljac svakog važnog softvera (u bilo kom programskom jeziku), pa ćemo istražiti različite aspekte funkcija u ovom poglavlju.
Funkcije definišemo pomoću ključne reči `def`. Posle ove reči je *identifikaciono* ime funkcije, iza koga slede zagrade koje mogu okruživati neka imena promenljivih, a na kraju stoji dvotačka, kojom se završava linija. Ispod sledi blok naredbi koje su deo ove funkcije. Primer će pokazati da je sve to, u suštini, veoma jednostavno:
Primer (sačuvati kao `funkcija1.py`):

```
def kaziZdravo():
    print('Pozdrav Svima!')  # blok koji pripada funkciji
# Kraj funkcije

kaziZdravo()  # poziva funkciju
kaziZdravo()  # opet poziva funkciju
```

Izlaz:

```
$ python3 funkcija1.py
Pozdrav Svima!
Pozdrav Svima!
```

Kako to funkcioniše:
Mi definišemo funkciju koja se zove `kaziZdravo` koristeći sintaksu kao što je objašnjeno u prethodnom pasusu. Ova funkcija ne uzima parametre i zato ne postoje promenljive deklarisane u zagradama. Parametri u funkcijama su samo ulaz za funkciju, tako da joj možemo "dati" različite vrednosti, za koje će nam funkcija vratiti odgovarajuće rezultate.
Primetimo da možemo pozvati istu funkciju dva puta što znači da ne morate ponovo da pišete istu kod.

9.1 Parametri funkcije

Funkcija može da "uzima" parametre, koji nisu ništa drugo nego vrednosti koje prosleđujemo toj funkciji, tako da funkcija može da *uradi* nešto nad podacima (ili bilo šta što poželimo) upotrebom tih vrednosti. Ovi parametri su slični kao i promenljive, osim što su vrednosti ovih varijabli definisane kada smo pozvali funkciju i poseduju već dodeljene vrednosti kada funkcija radi.
Definisanje parametara se vrši tako što ih navodimo u zagradama u definiciji funkcije, i odvajaju se zarezima. Kada pozivamo funkciju, mi joj dodeljujemo vrednosti parametara na isti način. Napomena za korišćenje terminologije - imena data u definiciji funkcije se zovu *parametri* dok vrednosti koje šaljemo u funkciju prilikom njenog poziva, se nazivaju *argumenti*.
Primer (sačuvati kao `funk_param.py`):

```
def prikaziMaks(a, b):
    if a > b:
```

```
        print(a, 'je maksimalno')
    elif a == b:
        print(a,'je jednako sa', b)
    else:
        print(b, 'je maksimalno')

prikaziMaks(3, 4)  # direktno saljemo funkciji konstantne vrednosti

x = 5
y = 7

prikaziMaks(x, y)  # saljemo varijable funkciji
```

Izlaz:

```
$ python3 funk_param.py
4 je maksimalno
7 je maksimalno
```

Kako to funkcioniše:
Ovde smo definisali funkciju koja se zove `prikaziMaks` koja koristi dva parametra, koji se zovu `a` i `b` . Mi smo saznali veći broj koristeći jednostavne `if..else` komande, a zatim i prikazali veći broj.
Prvi put kada smo pozvali funkciju `prikaziMaks` , mi smo direktno postavili brojeve kao argumente. U drugom slučaju, mi zovemo funkciju sa varijablama kao argumentima. `prikaziMaks(x, y)` će se tumačiti tako da će vrednost argumenta `x` biti dodeljena parametru `a`, dok će vrednost argumenta `y` biti dodeljena parametru `b` . Funkcija `prikaziMaks` funkcioniše na isti način u oba slučaja.

9.2 Lokalne promenljive

Kada definišete promenljivu unutar bloka funkcije, ta promenljiva nije povezana na bilo koji način sa drugim varijablama sa istim imenima koje koristimo van te funkcije - odnosno imena tih promenljivih su *lokalnog karaktera* - tj. koriste se samo u toj funkciji. Ovo se zove *scope* (delovanje) promenljive. Sve promenljive deluju unutar bloka u kom su definisane počevši od tačke definisanja imena te promenljive (varijable).
Primer (sačuvati kao `funk_lokal.py`):

```
x = 50

def funk(x):
    print('x je', x)
    x = 2
    print('Promenili smo lokalnu vrednost x na', x)

funk(x)
print('x je jos uvek', x)
```

Izlaz:

```
$ python3 funk_lockal.py
x je 50
Promenili smo lokalnu vrednost x na 2
x je jos uvek 50
```

Kako to funkcioniše:
Prvi put smo prikazali *vrednost* koja ima ime *x* sa prvom linijom unutar tela funkcije. Python je koristio vrednost parametra definisan u glavnom bloku programa, iznad definicije funkcije.
Zatim smo dodelili vrednost 2 za promenljivu `x` . Ime `x` je lokalno za našu funkciju. Dakle, kada smo promenili vrednost `x`-a u funkciji, vrednost `x`-a definisanog u glavnom bloku ostaje nepromenjena.
Sa poslednjom komandom `print`, mi smo prikazali vrednost `x`-a, i ona je ista kao što je definisano u glavnom bloku iznad poziva funkcije, čime se potvrđuje da nije bilo uticaja na njenu vrednost, od lokalne vrednosti koja je zadata u okviru prethodnog poziva funkcije.

9.3 Korišćenje globalnih promenljivih

Ako želite da dodelite vrednost imenu promenljive, koje je definisano na najvišem nivou programa (tj. nije u bilo kakvom delokruguod nekih stvari kao što su funkcija ili klasa), onda morate da kažete Python-u da to ime nije lokalno, već da je *globalno*. Mi to postižemo korišćenjem komande `global`. Nemoguće je dodeliti vrednost promenljivoj koja je definisana izvan funkcije bez `global` naredbe.
Vi možete da koristite vrednosti tih promenljivih definisanih izvan funkcije (pod pretpostavkom da ne postoji promenljiva sa istim imenom unutar funkcije). Međutim, to ne treba raditi i to treba izbegavati u širokom luku, jer bi bilo nejasno čitaocima programa, i izazvalo bi konfuziju oko toga gde je ta promenljiva u stvari definisana. Koristeći `global` komandu postaje dovoljno jasno da je promenljiva definisana u spoljašnjem bloku.
Primer (sačuvati kao `funk_global.py`):

```
x = 50

def funk():
    global x

    print('x je', x)
    x = 2
    print('Promenjena globalna vrednost promenljive x na', x)

funk()
print('Vrednost promenljive x sada je', x)
```

Izlaz:

```
$ python3 funk_global.py
x je 50
Promenjena globalna vrednost promenljive x na 2
Vrednost promenljive x sada je 2
```

Kako to funkcioniše:
`global` naredba se koristi da bi se naglasilo da je `x` globalna promenljiva - dakle, kada smo dodelili

neku vrednost promenljivoj `x` unutar funkcije, ta promena se odražava i kada koristimo vrednost `x` u glavnom bloku.
Možete da navedete više od jedne globalne promenljive koristeći istu `global` naredbu npr `global x, y, z` .

9.4 Podrazumevane vrednosti argumenata

U nekim funkcijama ćete želeti da neki parametri budu *opcionalni* i da koriste podrazumevane vrednosti u slučaju da korisnik ne želi da prosledi vrednosti za njih. To ćemo postići uz pomoć podrazumevanih vrednosti argumenata. Možete da odredite podrazumevane vrednosti argumenata za parametre dodavanjem imenu parametra unutar definicije funkcije operator dodele (=), a potom sledi podrazumevana vrednost.
Imajte na umu da podrazumevana vrednost argumenta mora da bude konstanta. Tačnije, podrazumevana vrednost treba da bude nepromenljiva - ovo je detaljnije objašnjeno u kasnijim poglavljima. Za sada, samo zapamtite ovo.
Primer (sačuvati kao `funk_podraz.py`):

```
def kazi(poruka, puta = 1):
    print(poruka * puta)

kazi('Pozdrav')
kazi('Svima', 5)
```

Izlaz:

```
$ python3 funk_podraz.py
Pozdrav
SvimaSvimaSvimaSvimaSvima
```

Kako to funkcioniše:
Funkcija sa nazivom `kazi` se koristi za prikaz stringa onoliko puta koliko navodimo. Ako mi ne navedemo vrednost, onda po defaultu, string će se prikazati samo jednom. Mi smo ovo postigli navođenjem podrazumevane vrednosti argumenta 1 za parametar `puta` .
U prvoj upotrebi funkcije `kazi` , mi joj prosleđujemo samo string i ona prikazuje string jednom. U drugoj upotrebi funkcije `kazi` , mi prosleđujemo kako string, tako i argument vrednosti 5, koji funkciji govori da želimo da poruka u stringu treba da bude *prikazana* 5 puta.
Važno
Samo onim parametrima koji su na kraju liste parametara možemo davati podrazumevane vrednosti argumenata, odnosno, ne možemo imati parametar sa podrazumevanom vrednošću argumenata koji se nalazi ispred parametra bez podrazumevane vrednosti argumenta u listi parametara prilikom definisanju funkcije.
To je zato što se vrednosti dodeljuju parametrima na osnovu njihovog položaja. Na primer, `def funk(a, b=5)` je ispravno, dok `def funk(a=5, b)` *nije*.

9.5 Argumenti definisani pomoću ključnih reči

Ako imate neke funkcije sa mnogo parametara i želite da definišete samo neke od njih, onda možete da date vrednosti za takve parametre tako što ćete ih imenovati (dati im ime prilikom definisanja funkcije)- što se zove definisanje *ključnih reči* argumenata. Poenta je da koristimo ime (ključnu reč) umesto pozicije (koju smo koristili do sad) da odredimo neki argument za potrebe funkcije.

U ovom slučaju postoje dve *prednosti* - jedna je da korišćenje funkcija mnogo lakše, jer ne moramo da brinemo o redosledu i poziciji argumenata. Druga je da možemo dati samo vrednosti onih parametara kojima želimo da damo, pod uslovom da ostali parametri imaju podrazumevane vrednosti argumenata.
Primer (sačuvati kao `funk_kljuc.py`):

```
def funk(a, b = 5, c = 10):
    print('a je', a, ',a b je', b, 'i c je', c)

funk(3,7)
funk(25, c = 24)
funk(c=50, a = 100)
```

Izlaz:

```
$ python3 funk_kljuc.py
a je 3 ,a b je 7 i c je 10
a je 25 ,a b je 5 i c je 24
a je 100 ,a b je 5 i c je 50
```

Kako to funkcioniše:
Funkcija nazvana `funk` ima jedan parametar bez vrednosti podrazumevanog argumenta, i dva parametra sa vrednostima podrazumevanih argumenata.
U prvoj upotrebi, `funk(3, 7)` , parametar `a` dobija vrednost 3 , parametar `b` dobija vrednost 7 i `c` dobija podrazumevanu vrednost 10 .
U drugoj upotrebi `funk(25, c=24)` , promenljiva `a` dobija vrednost 25 zbog položaja argumenta. Zatim parametar `c` dobija vrednost 24 zbog njegovog imenovanja, odnosno ključne reči argumenta. Promenljiva `b` dobija podrazumevanu vrednost 5 .
U trećoj upotrebi `funk(c=50, a=100)` , mi koristimo ključne reči argumenata za sve navedene vrednosti. Primetite da smo naveli vrednost za parametar `c` pre drugih , iako smo definisali prvo `a` pa `c` unutar definicije funkcije.

9.6 VarArgs parametri

Ponekad ćete možda želeti da definiše funkciju koja može da preuzme *bilo koji* broj parametara, ovo se može postići pomoću zvezda (asteriksa) (sačuvajte kao `total.py`):

```
def total(inicijal=5, *brojevi, **kljucnereci):
    brojanje = inicijal
    for broj in brojevi:
        brojanje += broj
    for rec in kljucnereci:
        brojanje +=kljucnereci[rec]
    return brojanje

print(total(10, 1, 2, 3, povrce = 50, voce = 100))
```

Izlaz:

```
$ python3 total.py
166
```

Kako to funkcioniše:
Kada smo definisali parametre sa zvezdicama, kao što su na primer `*param` , onda svi argumenti od te pozicije pa do kraja se prikupljaju kao tuple koji će se zvati 'param'.
Slično tome, kada smo definisali parametar sa duplom zvezdicom kao što je `**param` , onda svi ključni argumenti od te tačke do kraja se prikupljaju kao rečnik (dictionary) koji se zove 'param'.
Mi ćemo objasniti tuple i rečnike u kasnijem poglavlju.

9.7 Parametri samo sa ključnim rečima

Ako želimo da definišemo određene parametre koji bi bili dostupni samo preko ključnih reči, *ali ne i* preko njihove pozicije argumenata, oni mogu biti definisani posle parametra sa zvezdom (sačuvajte kao `samo_kljucne_reci.py`):

```
def total(inicijal=5, *brojevi, ekstra_broj):
    brojanje = inicijal
    for broj in brojevi:
        brojanje += broj
    brojanje += ekstra_broj
    print(brojanje)

total(10, 1, 2, 3, ekstra_broj=50)
total(10, 1, 2, 3)
# Izbacuje gresku jer nismo naveli vrednost za argument 'ekstra_broj'
```

Izlaz:

```
$ python3 samo_kljucne_reci.py
66
Traceback (most recent call last):
  File "samo_kljucne_reci.py", line 9, in <module>
    total(10, 1, 2, 3)
TypeError: total() missing 1 required keyword-only argument:
'ekstra_broj'
```

Kako to funkcioniše:
Definisanje parametra posle parametara sa zvezdicom, rezultira u argumentu samo sa ključnim rečima. Ako ovakvim argumentima nismo dodelili podrazumevanu vrednost prilikom pozivanja funkcije, funkcija će izbaciti grešku da joj nismo isporučili argument sa ključnim rečima, kao što se vidi u gornjem primeru.
Obratite pažnju na korišćenje oznake `+=` koja je prečica operatora, tako da umesto da pišemo `x = x + y` , možemo pisati `x += y` .
Ako želite da imate argumente koji su samo sa ključnim rečima, ali nema potrebe za parametrom sa zvezdicom, onda jednostavno koristite zvezdu bez upotrebe bilo kojeg imena, kao što je `def total(inicijal=5, *, ekstra_broj)` .

9.8 return komanda

`return` komanda se koristi da nešto *vratimo* iz funkcije, odnosno da izađemo iz funkcije. Mi, isto tako, opciono možemo da *vratimo neku vrednost* iz funkcije.
Primer (sačuvati kao `funk_return.py`):

```
def maksimum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'Brojevi su jednaki'
    else:
        return y

print(maksimum(2, 3))
```

Izlaz:

```
$ python3 funk_return.py
3
```

Kako to funkcioniše:
`maksimum` funkcija vraća maksimum parametara, u ovom slučaju brojeva koje šaljemo u funkciju. Ona koristi jednostavane `if..else` komande da pronađe veću vrednost, a zatim *vraća* tu vrednost.
Imajte na umu da `return` komanda koja nema nikakvu vrednost (napisana je kao da ne vraća ništa) je ekvivalent kao da je napisano `return None` . `None` je poseban tip podataka u Python-u koji predstavlja ništavilo (nema vrednosti - ili, drugim rečima, ništa). Na primer, koristi se da označi da promenljiva nema nikakvu vrednost (znači da ima vrednost `None`) .
Svaka funkcija implicitno (znači, i ako nije napisan kod unutar funkcije koji bi vraćao vrednost) sadrži `return None` komandu na kraju, osim ako ste napisali svoju `return` komandu. Možete to videti pokretanjem `print(nekaFunkcija())` gde funkcija `nekaFunkcija` ne koristi `return` naredbu kao što je:

```
def nekaFunkcija():
    pass
```

`pass` komanda se koristi u Python-u da kaže da je to prazan blok naredbi (u praksi: ne radi ništa, što može biti korisno u nekim situacijama).
Primetite
Postoji već ugrađena funkcija koja se zove `max` koja već radi "naći MAKSIMUM" funkcionalnost, tako da možete da koristite ovu ugrađenu funkciju kad god je to moguće i kada je to potrebno.

9.9 DocStrings (stringovi koji služe kao dokumentacija našeg programa)

Python ima jednu divnu funkciju koja se zove *string dokumentacije*, koja se obično naziva svojim kraćim imenom *DocStrings*. DocStrings su važno sredstvo koje bi trebalo da koristite, jer nam ono pomaže da bolje dokumentujemo program i čini ga lakše razumljivim. Zapravo, mi čak možemo da dobijemo DocStrings vraćen iz, recimo funkcije, u trenutku dok naš program zapravo radi!
Primer (sačuvati kao `funk_dok.py`):

```
def prikazMaksimuma(x, y):
    '''Prikazuje vecu vrednost dva broja.

    Ta dva broja moraju biti integeri (celi brojevi)'''
    x = int(x)  # konvertuje u integer, ukoliko je to moguce
    y = int(y)
```

```
    if x > y:
        print(x, 'je maksimum')
    else:
        print(y, 'je maksimum')

prikazMaksimuma(3, 5)
print(prikazMaksimuma.__doc__)
```

Izlaz:

```
$ python3 funk_dok.py
5 je maksimum
Prikazuje vecu vrednost dva broja.

    Ta dva broja moraju biti integeri (celi brojevi)
```

Kako to funkcioniše:
String na prvoj logičkoj liniji unutar funkcije je *DocString* za tu funkciju. Imajte na umu da se DocStrings mogu primenjivati i na modulima i klasama, koje ćemo učiti u odgovarajućim narednim poglavljima.
Konvencija koja je usvojena za DocString je da je to string iz više linija, gde prva linija počinje sa velikim slovom a završava sa tačkom. Zatim druga linija je prazna, propraćene detaljnim objašnjenjem počev od trećeg reda. *Čvrsto se savetuje* da pratite ovu konvenciju za sve vaše DocStrings unutar svih vaših važnih funkcija.
Možemo pristupiti DocString-u u `prikazMaksimuma` funkciji koristeći `__doc__` (primetiti *duplu donju crticu* ispred i iza) atribut (pripadajuće) funkcije. Zapamtite da Python tretira *sve* kao objekat a to uključuje i funkcije. Mi ćemo saznati više o objektima u poglavlju o klasama.
Ako ste koristili `help()` u Python-u, onda ste već videli korišćenje DocStrings-a! Ono što funkcija pomoći radi je samo da iščita `__doc__` atribut tražene funkcije, a zatim ga prikazuje na lep način za vas. Možete to da isprobate na funkciji iz primera - samo uključite `help(prikazMaksimuma)` u vaš program. Ne zaboravite da pritisnete taster q da biste izašli iz `help`-a .
Automatizovani alati mogu preuzeti dokumentaciju za vaš program na ovaj način. Zato ja *jako preporučujem* da koristite DocStrings za bilo koje ne-trivijalne funkcije koje pišete. `pydoc` komanda koja dolazi sa instalacijom Python-a radi slično `help()` komandi koja koristi DocStings.

9.10 Rezime

Videli smo mnoge osobine funkcija, ali imajte na umu da mi još uvek nismo pokrili sve njihove aspekte. U svakom slučaju, mi smo pokrili većinu onoga što ćete koristiti u vezi sa Python funkcijijama svakodnevno.
U sledećem poglavlju ćemo videti kako se koriste, kao i kako se kreiraju moduli u Python-u.

10 Moduli

Vi ste videli kako možete više puta koristiti svoj kod u vašem programu kada samo jednom definišete funkciju. Šta ako bi ste želeli da ponovo koristite niz funkcija u drugim programima koje ste napisali? Kao što ste možda pogodili, odgovor je u modulima.
Postoje različite metode pisanja modula, ali najjednostavniji način je da napravite datoteku sa ekstenzijom `.py` koja bi sadržala funkcije i promenljive koje bi trebale našem programu.
Druga metoda je da se napiše modul na nativnom jeziku na kojem je i pisan Python interpreter. Na primer, možete da napišete modul u C programskom jeziku i kada ga kompajlirate, on može da se koristi u vašem Python kodu, kada koristite standardni Python interpreter.
Modul može da bude *korišćen* i od strane nekog drugog programa, kako bi taj program koristio funkcionalnosti iz modula. Takođe, to je način na koji koristimo Python-ovu standardnu biblioteku. Prvo ćemo videti kako da koristite module standardnih biblioteka.
Primer (sačuvati kao `koristiti_sys.py`):

```
import sys

print('Argumenti komandne linije su: ')
for i in sys.argv:
    print(i)

print('\n\nPYTHONPATH je', sys.path, '\n')
```

Izlaz:

```
$ python3 koristiti_sys.py mi smo argumenti
Argumenti komandne linije su:
koristiti_sys.py
mi
smo
argumenti


PYTHONPATH je ['C:\\py', 'C:\\WINDOWS\\system32\\python33.zip',
'C:\\Python33\\DLLs', 'C:\\Python33\\lib', 'C:\\Python33',
'C:\\Python33\\lib\\site-packages']
```

Kako to funkcioniše:
Prvo, mi smo *uvezli* `sys` modul pomoću `import` naredbe. U suštini, to za nas znači da govorimo Python-u da želimo da koristimo ovaj modul. `sys` modul sadrži funkcionalnosti vezane za Python-ov interpreter i njegovu okolinu, odnosno *sistem*.
Kada Python izvrši `import sys` komandu, on u stvari traži `sys` modul. U ovom slučaju, to je jedan od ugrađenih modula, a samim tim i Python zna gde da ga nađe.
Ako to nije bio kompajliran modul tj. modul napisan u Python-u, tada će Python-ov interpreter tražiti taj modul u direktorijumima navedenim u `sys.path` promenljivoj. Ako je modul pronađen, komande koje se nalaze u telu tog modula se pokreću i modul postaje *dostupan* za korišćenje. Imajte na umu da se inicijalizacija modula vrši samo *prvi* put kada smo uvezli modul u program.
`argv` promenljivoj u `sys` modulu se pristupa tako što koristimo oznaku tačka odnosno `sys.argv` .

To jasno ukazuje da je ovo ime promenljive deo `sys` modula. Još jedna prednost ovog pristupa je da se ime ne kosi sa nekom drugom `argv` promenljivom koju bi koristili u našem programu.
`sys.argv` promenljiva je *lista* stringova (liste su detaljno objašnjene u kasnijem poglavlju). Konkretno, `sys.argv` sadrži listu *argumenata komandne linije*, odnosno argumenata koje smo postavili za svoj program koristeći komandnu liniju (pogledati primer, ukoliko se do sada niste susretali sa pokretanjem programa iz komandne linije).
Ako koristite neki IDE program za pisanje i pokretanje Python programa, nađite način da dodelite argumente komandne linije pri pokretanju Vašeg programa (gledajte u menijima IDE-a).
Ovde, kad smo izvršiti `python3 koristiti_sys.py mi smo argumenti` , mi pokrećemo modul `koristiti_sys.py` sa `python3` komandom, a ostale stvari koje smo napisali su argumenti koje prosleđujemo programu. Python čuva argumente komandne linije u `sys.argv` promenljivoj za nas da ih po potrebi koristimo.
Zapamtite, ime skripte koja se trenutno izvršava je uvek prvi argument u `sys.argv` listi. Dakle, u ovom slučaju ćemo imati '`koristiti_sys.py`' kao `sys.argv[0]` , '`mi`' kao `sys.argv[1]` , '`smo`' kao `sys.argv[2]` i '`argumenti`' kao `sys.argv[3]` . Primetite da Python počinje brojanje od 0, a ne od 1.
`sys.path` promenljiva sadrži spisak imena direktorijuma iz kojih se uvoze moduli. Primetite da je prvi string u `sys.path` prazan - ovaj prazan string ukazuje da je trenutni direktorijum takođe deo `sys.path` koji je isti kao i `PYTHONPATH` promenljiva okruženja. To znači da možete direktno uvoziti module koji se nalaze u tekućem direktorijumu. U suprotnom, moraćete da postavite modul u jedan od direktorijuma navedenih u `sys.path` .
Imajte na umu da trenutni direktorijum je direktorijum iz koga je pokrenut program. Pokreni `import os; print(os.getcwd())` da bi saznao trenutni direktorijum vašeg programa.

10.1 Kompajlirani .pyc fajlovi

Uvoz modula je relativno skupa stvar, pa Python radi neke trikove da bi se cela stvar izvršavala brže. Jedan od načina je stvaranje *bajt-kompajliranih* fajlova sa ekstenzijom `.pyc`, koji su neki srednji oblik u koji Python pretvara neki program (sećate li se uvodnog dela knjige o načinu na koji Python radi?). Ovaj `.pyc` fajl je koristan kada uvozite taj modul sledeći put iz drugog programa - sve to će se odvijati mnogo brže, jer je veći deo potrebne obrade prilikom uvoza modula urađen. Takođe, ovi bajt-kompajlirani fajlovi su nezavisni od platforme.
Primetiti
Ovi `.pyc` fajlovi se obično kreirani u istom direktorijumu kao i odgovarajući `.py` fajlovi. Ako Python nema dozvole da kreira fajlove u tom folderu, onda `.pyc` fajl neće biti kreiran.

10.2 from . . . import komanda

Ako bi ste želeli da direktno uvezete `argv` promenljivu u program iz prethodnog primera (da bi se izbeglo da kucate `sys.` svaki put kada je pozivate), onda možete da koristite naredbu `from sys import argv` .
U principu, *trebalo bi da izbegavate* ovakvu komandu i umesto nje koristite `import` komandu, jer će vaš program izbeći sukobe imena (ukoliko se pojavi još jedna promenljiva sa istim imenom - npr. uvezena iz nekog drugog modula), a i program će biti čitljiviji.
Primer:

```
from math import sqrt
print("Kvadratni koren iz 16 je" , sqrt( 16 ))
```

10.3 Imena modula

Svaki modul ima svoje ime i naredbe u modulu mogu saznati ime svog modula. Ovo je zgodno za određene namene kada treba saznati da li se modul pokreće samostalno ili je uvežen u neki program. Kao što je pomenuto ranije, kada se modul uvozi po prvi put, njegov kod se izvršava. Možemo iskoristiti ovo da napravimo modul koji će se ponašati na različite načine u zavisnosti od toga da li se koristi sam za sebe ili se uvozi iz drugog modula. To se može postići korišćenjem `__name__` atributa modula.
Primer (sačuvati kao `koristiti_ime.py`):

```
if __name__ == '__main__':
    print('Program se pokrece sam')
else:
    print('Program se pokrece iz drugog programa')
```

Izlaz:

```
$ python3 koristiti_ime.py
Program se pokrece sam
$ python3
>>> import koristiti_ime
Program se pokrece iz drugog programa
>>>
```

(ne zaboravite da promenite direktorijum u onaj u kom Vam se nalazi ovaj modul!)
Kako to funkcioniše:
Svaki modul u Python-u ima definisano njegovo `__name__`. Ako je ono `'__main__'` , podrazumeva se da je modul pokrenut samostalno od strane korisnika pa možemo preduzeti odgovarajuće akcije.

10.4 Izrada sopstvenih modula

Kreiranje sopstvenih modula je lako, zapravo, to smo radili sve vreme! To je zato što je svaki Python program takođe i modul. Vi samo treba da proverite da li ima `.py` ekstenziju. Sledeći primer bi trebalo da malo razjasni ovo.
Primer (sačuvati kao `mojmodul.py`):

```
def kaziZdravo():
    print('Zdravo! Ovo govori mojmodul.')

__version__ = '0.1'
```

Ovo gore je primer *modula*. Kao što možete da vidite, ne postoji ništa posebno niti specijalno unutar njega u odnosu na naše dosadašnje Python programe. Sledeća što ćete videti je kako da koristite ovaj modul u našim drugim Python programima.
Zapamtite da modul treba biti smešteni u istom direktorijumu kao i program u koji ga uvozimo, ili u jednom od direktorijuma navedenih u `sys.path` .
Drugi modul (sačuvajte kao `mojmodul_demo.py`):

```
import mojmodul

mojmodul.kaziZdravo()
```

```
print('Verzija', mojmodul.__version__)
```

Izlaz:

```
$ python3 mojmodul_demo.py
Zdravo! Ovo govori mojmodul.
Verzija 0.1
```

Kako to funkcioniše:
Primetite da koristimo isti način označavanja pomoću tačaka za pristup članovima modula. Python ima dobru osobinu koja je ponavljanje upotrebe istih metoda označavanja, što mu i daje karakterističan "Pythonic" osećaj prilikom programiranja, jer ne moramo da učimo neke nove načine da radimo različite stvari.
Evo je verzija koja koristi `from..import` sintaksu (sačuvajte kao `mojmodul_demo2.py`):

```
from mojmodul import kaziZdravo, __version__

kaziZdravo()
print('Verzija', __version__)
```

Izlaz `mojmodul_demo2.py` je isti kao izlaz `mojmodul_demo.py` .
Primetimo da, ako je već `__version__` ime definisano u modulu koji uvozi `mojmodul`, došlo bi do sukoba između tih imena. Mogućnost da se to desi je vrlo velika, zato što je uobičajena praksa da se za svaki modul proglasi neki broj koji predstavlja njegovu verziju, koristeći upravo ovo ime. Dakle, uvek se preporučuje da koristite `import` komandu iako ona može učiniti vaš program malko dužim.
Takođe možete da koristite:

```
from mojmodul import *
```

Ovo će uvesti sva javna imena kao što su `kaziZdravo` ali neće uvoziti `__version__` jer počinje sa duplom donjom crtom.
Zen Python-a
Jedan od vodećih principa Python-a je da je "Eksplicitno bolje nego Implicitno". Pokrenite `import this` da naučite više i pogledajte ovu diskusiju na sajtu StackOverflow u kojoj se navode primeri za svaki od principa.

10.5 Funkcija dir

Možete da koristite ugrađenu `dir` funkciju da izlistate identifikatore koje je definisao objekat. Na primer, za neki modul identifikatori uključuju funkcije, klase i promenljive definisane u tom modulu.
Kada navedemo ime modula u `dir()` funkciju, ona vraća listu imena definisanih u tom (navedenom) modulu. Ako mu ne imenujemo nijedan argument, vraća listu imena definisanih u tekućem modulu.
Primer:

```
$ python3
>>> import sys # daje listu atributa, u ovom slucaju, za sys modul
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__',
```

```
'__loader__','__name__', '__package__', '__stderr__', '__stdin__',
'__stdout__', '_clear_type_cache', '_current_frames', '_getframe',
'api_version', 'argv', 'builtin_module_names', ' byteorder',
'call_tracing', 'callstats', 'copyright', 'displayhook', 'dllhandle' ,
'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix',
'executable', 'exit', 'flags', 'float_info', 'getcheckinterval',
'getdefaultencoding', 'getfil esystemencoding', 'getprofile',
'getrecursionlimit', 'getrefcount', 'getsizeof', 'gettrace',
'getwindowsversion', 'hexversion', 'intern', 'maxsize', 'maxunicode ',
'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache',
'platfor m', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setprofile',
'setrecursionlimit ', 'settrace', 'stderr', 'stdin', 'stdout',
'subversion', 'version', 'version_in fo', 'warnoptions', 'winver']
>>> dir() # daje listu atributa za tekuci modul
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
>>> a = 5 # kreira novu variablu 'a'
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a', 'sys']
>>> del a # uklanja/brise ime
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
>>>
```

Kako to funkcioniše:
Prvo, vidimo korišćenje `dir` funkcije na uvezen `sys` modul. Možemo videti ogroman spisak atributa koje modul sadrži.
Dalje, mi koristimo `dir` funkciju bez unošenja parametre u nju. Po defaultu, vraća listu atributa za trenutni modul. Primetimo da je u listi modul koji smo uvezli (`sys`), tj on je takođe deo ovog spiska.
U cilju da prkažemo `dir` u akciji, mi smo definisali novu promenljivu `a` i dodeliti joj vrednost, a zatim proverili `dir` i vidimo da sad postoji dodatna vrednost u listi sa istim imenom kao i nova promenljiva. Mi smo uklonili promenljivu/atribut tekućeg modula pomoću `del` komande i promenu ponovo primećujemo u izlazu `dir` funkcije.
Napomena o `del` - ova komanda se koristi za *brisanje* promenljivih/imena promenljivih, i posle izvršenja komande, u ovom slučaju `del a` , više ne možete da pristupite promenljivaoj `a` - to je kao da nikad nije ni postojala uopšte.
Imajte na umu da je `dir()` funkciju možete primeniti na *bilo koji* objekat. Na primer, pokrenite `dir('print')` da se upoznate sa atributima prikaza, ili `dir(str)` za atribute klase stringova.

10.6 Pakovanje

Do sada, verovatno ste počeli da razmišljate oko hijerarhije i načina organizovanja svojih programa. Promenljive obično idu unutar funkcija. Funkcije i globalne promenljive obično idu unutar modula. Šta ako ste želeli da organizujete module? To je mesto gde pakovanja (packages) dolaze u prvi plan. Pakovanja su samo folderi modula sa posebnom `__init__.py` datotekom koja ukazuje Python-u da je ova fascikla posebna, jer sadrži Python module.
Recimo da želite da kreirate paket pod nazivom 'svet' koji u sebi ima pod-pakete sa nazivima 'Aziji', 'Afrika' itd, i ovih sub paketi redom sadrže module poput 'Indija', 'Madagaskar', itd.

Ovo je kako bi vi trebali da napravite strukturu direktorijuma:

```
- <neki folder koji se nalazi unutar sys.path>/
    - svet/
        - __init__.py
        - azija/
            - __init__.py
            - indija/
                - __init__.py
                - program.py
        - afrika/
            - __init__.py
            - madagaskar/
                - __init__.py
                - drugiProgram.py
```

Pakovanje je samo konvencija o hijerarhijskom organizovanju modula. Vi ćete videti mnoge primere ovoga u standardnim bibliotekama.

10.7 Rezime

Baš kao što su funkcije delovi programa koji se mogu više puta primenjivati unutar tog programa, tako su i moduli programi koji se mogu više puta upotrebljavati unutar drugih programa. Pakovanja su ništa drugo do hijerarhija za organizovanje modula. Standardna biblioteka koja dolazi uz Python je primer takvog skupa paketa i modula.
Videli smo u ovom poglavlju, kako se koriste moduli i kako se stvaraju sopstveni moduli.
U sledećem poglavlju ćemo učiti o nekim zanimljivim konceptima koji se zovu strukture podataka.

11 Strukture podataka

Strukture podataka su u osnovi samo to - *strukture* koje mogu da drže neke *podatke* na okupu. Drugim rečima, oni se koriste za skladištenje kolekcije srodnih podataka.
Postoje četiri ugrađene strukture podataka u Python-u - liste, tuple, rečnici (dictionary) i skupovi (setovi). Mi ćemo videti kako se koristi svaki od njih i kako će nam oni učiniti život mnogo lakšim.

11.1 Liste

Liste su strukture podataka koje sadrže kolekciju nekih predmeta u nekom određenom rasporedu, odnosno vi možete da čuvate *niz* stavki unutar liste. Možete to lako sebi predstaviti ako zamislite spisak za kupovinu, gde imate listu stavki za kupovinu, osim što ćete verovatno imate svaku stavku u posebnom redu u toj vašoj listi za kupovinu, dok se u Python-u stavlja zarez između njih.
Spisak stavki treba da bude u uglastim zagradama, tako da Python razume da mu vi govorita da je u pitanju lista. Kada ste napravili listu, možete da dodate, uklonite ili tražite stavke unutar liste. Pošto možemo dodavati i uklanjati stavke, mi kažemo da je lista *promenljiv* (mutable) tip podataka, odnosno ovaj tip može biti menjan.

11.1.1 Brzo predstavljanje objekata i klasa

Iako sam, generalno, odlagao objašnjavanje objekata i klasa do sada, ipak je malo objašnjenje potrebno, tako da možete bolje da razumete liste. Mi ćemo istražiti ovu temu detaljnije u kasnijem poglavlju.
Lista je primer korišćenja objekata i klasa. Kada koristimo promenljivu `i` i kada joj dodelimo vrednost, recimo dodamo joj celobrojno 5, možete misliti o tome kao o stvaranju **objekta** (tj. instance) `i` koja je **klase** (tj. tipa) `int` (celobrojne vrednosti). U suštini, možete pročitati `help(int)` da bi ovo malo bolje razumeli.
Klasa takođe može imati **metode**, odnosno funkcije definisane samo za korišćenje u toj klasi. Vi možete koristiti te funkcionalnosti samo kada imate objekat te klase. Na primer, Python poseduje `append` metod za klasu `list`, koja vam omogućava da dodate stavku na kraju liste. Na primer, `mojalista.append('neka stvar')` će dodati string `'neka stvar'` u listu `mojalista` . Obratite pažnju na upotrebu tačke kao oznake za pristup metodama nekog objekata.
Klasa može imati **polja** (field), koja nisu ništa drugo nego promenljive definisane za korišćenje samo u toj klasi. Možete koristiti ove promenljive/imena samo u slučaju kada imate neki objekat iz te klase. Poljima se takođe pristupa oznakom sa tačkom, na primer, `mojalista.polje` .
Primer (sačuvati kao `koristiti_liste.py`):

```
# Ovo je moj spisak za kupovinu
spisakkupovine = ['jabuka', 'mango', 'sargarepa', 'banana']

print('Ja imam', len(spisakkupovine), 'stvari da kupim.')

print('Te stvari su:', end = ' ')
for stvar in spisakkupovine:
    print(stvar, end = ', ')

print('\nJa takodje treba da kupim i pirinac.')
spisakkupovine.append('pirinac')
print('Moj spisak za kupovinu je sad', spisakkupovine)
```

```
print('Sada cu da sortiram moj spisak.')
spisakkupovine.sort()
print('Sortirana lista je', spisakkupovine)

print('Prva stvar koju cu kupiti je', spisakkupovine[0])
starastvar = spisakkupovine[0]

del spisakkupovine[0]
print('Kupio sam', starastvar)
print('Moj spisak za kupovinu je sada', spisakkupovine)
```

Izlaz:

```
$ python3 koristiti_liste.py
Ja imam 4 stvari da kupim.
Te stvari su: jabuka, mango, sargarepa, banana,
Ja takodje treba da kupim i pirinac.
Moj spisak za kupovinu je sad ['jabuka', 'mango', 'sargarepa',
'banana', 'pirinac']
Sada cu da sortiram moj spisak.
Sortirana lista je ['banana', 'jabuka', 'mango', 'pirinac',
'sargarepa']
Prva stvar koju cu kupiti je banana
Kupio sam banana
Moj spisak za kupovinu je sada ['jabuka', 'mango', 'pirinac',
'sargarepa']
```

Kako to funkcioniše:

Promenljiva `spisakkupovine` je šoping lista za nekoga ko će da ide u prodavnicu. U `spisakkupovine`, mi čuvamo niz imena stavki za kupiti, ali zapamtite da *možete dodati bilo kakav objekat* u listu, uključujući brojeve, pa čak i druge liste.

Takođe smo koristili `for..in` petlju da bi smo prelazili preko stavke na listi. Do sada, mora da ste shvatili da je lista takođe niz (sekvenca). O specijalnostima sekvenci ćemo govoriti u kasnijem delu.

Obratite pažnju da koristimo `end` ključnu reč argumenta za `print` funkciju, što joj ukazuje na to da želimo da se izlazna linija završava sa razmakom umesto uobičajenog novog reda.

Dalje, mi smo dodali stavku na listi koristeći `append` metod u listu predmeta, kao što je već razmatrano ranije. Zatim smo proverili da li je stavka zaista dodata u listu, prikazivanjem sadržaja liste jednostavnim prosleđivanjem liste u `print` naredbu koja ja lepo prikaže.

Onda smo sortirali listu pomoću `sort` metode liste. Važno je da se shvati da ovaj metod utiče na samu listu i ne vraća (return) modifikovanu listu - već tu listu zaista i menja! - ovo se bitno razlikuje od načina rada sa stringovima. To je upravo ono, na šta mislimo kada kažemo da su liste *promenljive* a da su stringovi *nepromenljivi*.

Dalje, kad smo završite kupovinu stavke u prodavnici, želimo da je uklonimo iz liste. Mi smo to postigli pomoću `del` komande. Mi ćemo napisati koju stavku sa liste želimo da uklonimo i `del` komanda to odradi za nas. Mi smo odredili da želimo da ukloni prvu stavku sa liste i time koristimo `del spisakkupovine[0]` (sećate se toga da Python počinje brojanje od 0).

Ako želite da znate sve metode definisane objektom liste, pogledajte `help(list)` za detalje.

11.2 Tuple

Tuple se koriste za držanje više objekata zajedno, na jednom mestu. Mislite o njima kao da su slične listama, ali bez mnogih funkcionalnosti koje klasa lista pruža. Jedna od glavnih karakteristika tuple je da su one **nepromenljive** kao i stringovi, odnosno ne možete da modifikuje podatke.
Tuple se definišu tako što navodimo stavke razdvojene zarezima unutar opcionih zagrada ().
Tuple se obično koriste u slučajevima kada naredba ili korisnički-definisana funkcija može bezbedno da pretpostavi da kolekcija vrednosti, odnosno, da se tuple korišćenih vrednosti neće promeniti.
Primer (sačuvati kao `koristiti_tuple.py`):

```
zoo = ('piton', 'slon', 'pingvin') # zapamtite da su zagrade opcionalne
print('Broj zivotinja u zooloskom vrtu je', len(zoo))

novi_zoo = 'majmin', 'kamila', zoo
print('Broj kaveza u novom zooloskom vrtu je', len(novi_zoo))
print('Sve zivotinje u novom zooloskom vrtu su', novi_zoo)
print('Zivotinje koje su donete iz starog zooloskog vrta su',
novi_zoo[2])
print('Poslednja zivotinja doneta iz starog zooloskog vrta je',
novi_zoo[2][2])
print('Broj zivotinja u novom zooloskom vrtu je', len(novi_zoo)-
1+len(novi_zoo[2]))
```

Izlaz:

```
$ python3 koristiti_tuple.py
Broj zivotinja u zooloskom vrtu je 3
Broj kaveza u novom zooloskom vrtu je 3
Sve zivotinje u novom zooloskom vrtu su ('majmin', 'kamila', ('piton',
'slon', 'pingvin'))
Zivotinje koje su donete iz starog zooloskog vrta su ('piton', 'slon',
'pingvin')
Poslednja zivotinja doneta iz starog zooloskog vrta je pingvin
Broj zivotinja u novom zooloskom vrtu je 5
```

Kako to funkcioniše:
Promenljiva `zoo` se odnosi na tuplu nekih stavki. Vidimo da `len` funkcija može da se koristi da bi se dobila dužina tuple. Ovo takođe ukazuje na to da je takođe i tupla sekvenca.
Mi sada selimo ove životinje u novi zoološki vrt, jer se stari zoo vrt zatvara. Dakle, `novi_zoo` tupla sadrži neke životinje koje su već tu, zajedno sa životinjama koje smo doveli iz starog zoološkog vrta. Da se vratimo u realnost, imajte na umu da tuple unutar neke druge tuple ne gube svoj identitet (tj i dalje su - tuple, a ne pojedinačni objekti koji su sačinjavali staru tuplu).
Možemo pristupiti stavkama unutar tuple navodeći položaj te stavke u paru uglastih zagrada baš kao što smo uradili za liste. Ovo se zove operator *indeksiranja*. Mi pristupamo trećoj stavki u `novi_zoo` navodeći `novi_zoo[2]`, a da bi smo pristupili trećoj stavki u okviru treće stavke u `novi_zoo` tupli navodimo `novi_zoo[2][2]` . Ovo je prilično jednostavna kada shvatite poentu.
Zagrada '()'
Iako su zagrade opcioni element tuple, uvek bi trebalo da ih navodimo da bi postalo očigledno da je to tupla, pogotovo zato da bi se izbegavala dvosmislenost. Na primer, `print(1,2,3)` i

`print( (1,2,3) )` su dve različite stvari - prvi prikazuje tri broja, dok druga prikazuje tuple (koji sadrži tri broja).

Tuple sa 0 ili sa 1 stavkom

Prazna tupla se konstruiše sa praznim zagradama, kao što je npr. `mojaprazna = ()` . Međutim, za tuple sa samo jednom stavkom to nije tako jednostavno. Morate je navesti koristeći zarez posle prve (i jedine) stavke, tako da Python može razlikovati tuple i par zagrada koje okružuju objekat u nekom izrazu, odnosno morate da odredite `singlica = (2 , )` , ako želite tuplu koja sadrži samo stavku 2 .

Napomena za Perl programere

Liste unutar liste ne gube svoje osobine, odnosno, liste neće biti "izravnane" kao u Perlu. Isto važi i za tuple unutar tuple, ili tupla unutar liste, ili listu unutar tuple, itd. Što se Python-a tiče, oni su samo objekti koji se čuvaju unutar drugog objekta, i to je sve.

11.3 Rečnik (Dictionary)

Rečnik je sličan adresaru, u kom možete naći adrese ili detalje o kontaktu osobe znajući samo njegovo/njeno ime, odnosno povezujemo **ključ** (ime) sa **vrednostima** (detalji). Imajte na umu da ključ mora biti jedinstven baš kao što se ne može saznati tačna informacija ako imate dve osobe sa totalno istim imenom.

Zapamtite da možete da koristite samo nepromenljive objekte (kao stringove) za ključeve u rečniku, ali možete da koristite bilo nepromenljive ili promenljive objekate za vrednosti u rečniku. Ovo u suštini znači da treba da koristite samo jednostavne objekte za ključeve u rečniku.

Parovi ključeva i vrednosti se definišu u rečniku pomoću notacije `d = {kljuc1 : vrednost1, kljuc2 : vrednost2 }` . Obratite pažnju da su ključ-vrednost parovi razdvojeni dvotačkom, a međusobno su odvojeni zarezima, a sve ovo je zatvoren u par vitičastih zagrada.

Zapamtite da ključ-vrednost parovi u rečniku nisu razvrstani ni na koji način. Ako želite da dobijete određeni poredak, onda ćete morati da ih sortirate pre nego što ih budete koristili.

Rečnici koje ćete koristiti su instance/objekti unutar `dict` klase.

Primer (sačuvati kao `koristiti_recnik.py`):

```
# 'ab' je skracenica za 'a'-address 'b'-book
ab = {'Pandimenzionalni' : 'pd@ma.cme',
      'mega'             : 'mno@hoo.moo',
      'giga'             : 'kokos@kokoda.lom',
      'troler'           : 'gong@pong.pet'}
print("Trolova adresa je", ab['troler'])

# brisanje para kljuc-vrednost
del ab['troler']

print('\nImamo {0} kontakata u adresaru\n'.format(len(ab)))

for ime, adresa in ab.items():
    print('Kontaktiraj {0} na {1}'.format(ime, adresa))

# Dodavanje para kljuc-vrednost
ab['Genije'] = 'popara@lebac.voda'
```

```
if 'Genije' in ab:
    print('\nAdresa genija je', ab['Genije'])
```

Izlaz:

```
$ python3 koristiti_recnik.py
Trolova adresa je gong@pong.pet

Imamo 3 kontakata u adresaru

Kontaktiraj Pandimenzionalni na pd@ma.cme
Kontaktiraj mega na mno@hoo.moo
Kontaktiraj giga na kokos@kokoda.lom

Adresa genija je popara@lebac.voda
```

Kako to funkcioniše:
Mi stvaramo rečnik `ab` koristeći notaciju koju smo već razmotrili. Zatim smo pristupili parovima ključ-vrednost navođenjem ključa pomoću operatora indeksiranja kao što je objašnjeno u kontekstu liste i tuple. Obratite pažnju na jednostavnu sintaksu.
Možemo obrisati par ključ-vrednost koristeći našeg starog prijatelja - `del` komandu. Mi smo jednostavno naveli rečnik i ključ kao vrednost za operator indeksiranja za par koji želimo da se ukloni i navodimo ga u `del` komandi. Nema potrebe da se zna vrednost koja odgovara ključu za ovu operaciju brisanja.
Dalje, mi smo pristupili svakom paru ključ-vrednost unutar rečnika korišćenjem `items` metode rečnika koji nam vraća listu tupli gde svaka tupla sadrži par stavki - ključ praćen vrednosti. Mi preuzmemo ovaj par i dodelimo ga varijablama `ime` i `adresa` odgovarajućim navedenim redosledom. Za svaki taj par koristimo `for..in` petlju i onda prikazujemo ove vrednosti u `for`-bloku.
Možemo dodati nove parove ključ-vrednost jednostavno pomoću operatora indeksiranja koji pristupa ključu i dodeljuje mu vrednost, kao što smo uradili za Genija u navedenom primeru.
Možemo da proverimo da li postoji par ključ-vrednost u nekom rečniku koristeći `in` naredbu.
Za listu metodama `dict` klase, pogledajte `help(dict)` .
Argumenti ključnih reči i rečnici
Jedno drugo razmišljanje, ako ste koristili argumente ključnih reči u svojim funkcijama, onda ste već koristili rečnike! Samo razmislite o tome - to je par ključ-vrednost koji ste definisali u listi parametara pri definiciji funkcije i kada pristupate promenljivoj u okviru vaše funkcije, navodite samo ključ koji pristupa rečniku (što se zove *simbol-tabela* u terminologiji dizajna kompajlera).

11.4 Sekvence

Liste, tuple i stringovi su primeri sekvenci, ali šta su sekvence (nizovi) i šta je to tako posebno u vezi sa njima?
Glavne karakteristike sekvenci su **testovi članstva**, (tj. `in` i `not in` ekspresije) i **operacija indeksiranja**, koje nam omogućavaju da direktno pristupimo određenoj stavki unutar neke sekvence.
Sve tri vrste pomenutih sekvenci - liste, tuple i stringovi, takođe imaju operaciju **isecanja** (slicing) koja nam omogućava da preuzmemo parče (isečak) niza, odnosno deo sekvence.
Primer (sačuvati kao `sekvenca.py`):

```
listakupovine = ['jabuka', 'mango', 'sargarepa', 'banana']
ime = 'swaroop'

# Operacija indeksiranja ili Subscription
print('Stvar 0 je', listakupovine[0])
print('Stvar 1 je', listakupovine[1])
print('Stvar 2 je', listakupovine[2])
print('Stvar 3 je', listakupovine[3])

print('Stvar -1 je', listakupovine[-1])
print('Stvar -2 je', listakupovine[-2])
print('Znak 0 je', ime[0])

# secenje liste
print('Stvari od 1 do 3 su', listakupovine[1:3])
print('Stvari od 2 do kraja su', listakupovine[2:])
print('Stvari od 1 do -1 su', listakupovine[1:-1])
print('Stvari od pocetka do kraja su', listakupovine[:])

# secenje stringa
print('Znakovi od 1 do 3 su', ime[1:3])
print('Znakovi od 2 do kraja su', ime[2:])
print('Znakovi od 1 do -1 su', ime[1:-1])
print('Znakovi od pocetka do kraja su', ime[:])
```

Izlaz:

```
$ python3 sekvenca.py
Stvar 0 je jabuka
Stvar 1 je mango
Stvar 2 je sargarepa
Stvar 3 je banana
Stvar -1 je banana
Stvar -2 je sargarepa
Znak 0 je s
Stvari od 1 do 3 su ['mango', 'sargarepa']
Stvari od 2 do kraja su ['sargarepa', 'banana']
Stvari od 1 do -1 su ['mango', 'sargarepa']
Stvari od pocetka do kraja su ['jabuka', 'mango', 'sargarepa',
'banana']
Znakovi od 1 do 3 su wa
Znakovi od 2 do kraja su aroop
Znakovi od 1 do -1 su waroo
Znakovi od pocetka do kraja su swaroop
```

Kako to funkcioniše:

Prvo, vidimo kako se koriste indeksi da se dobiju pojedinačne stavke iz sekvence. Ovo se takođe naziva i *operacija subskripcije*. Kada god ste naveli broj u sekvenci u uglastim zagradama, kao što

je prikazano gore, Python vam dostavlja stavku koja odgovara toj poziciji u sekvenci (nizu). Zapamtite da Python počinje sa brojanjem brojeva od 0. Dakle, `listakupovine[0]` je prva stavka, a `listakupovine[3]` četvrta stavka u `listakupovine` nizu.
Indeks može biti negativan broj, u kom slučaju, pozicija objekta se izračunava od kraja niza. Dakle, `listakupovine[-1]` se odnosi na poslednju stavku u nizu, a `listakupovine[-2]` na predposlednju stavku u nizu.
Operacija isecanja se upotrebljava navođenjem imena sekvence za kojim sledi opcioni par brojeva razdvojenih dvotačkom u uglastim zagradama. Imajte na umu da je ovo veoma slično operaciji indeksiranja koju ste koristili do sada. Zapamtite: brojevi su neobavezni ali dvotačka je obavezna.
Prvi broj (pre dvotačke) u operaciji isecanja, se odnosi na poziciju odakle isečak počinje, a drugi broj (posle dvotačke) pokazuje gde će se zaustaviti sečenje. Ako prvi broj nije naveden, Python će početi isecanje od početka niza. Ako je drugi broj izostavljen, Python će se zaustaviti na kraju niza. Imajte na umu da deo koji dobijamo *počinje* na startnoj poziciji i da *će se završiti* ispred krajnjeg navedenog položaja, odnosno početna pozicija je uključena, ali krajnja pozicija će da bude isključena iz odsečenog dela niza.
Tako `listakupovine[1:3]` daje parče niza sa početkom u poziciji 1, uključuje položaj 2, ali se zaustavlja na poziciji 3 i samim tim nam se vraća *parče* od dve stavke. Slično tome, `listakupovine[:]` vraća kopiju cele sekvence.
Takođe možete da radite odsecanje sa negativnim brojevima. Negativni brojevi se koriste da označe pozicije od kraja niza. Na primer, `listakupovine[:-1]` daje deo niza koji isključuje poslednju stavku u nizu, ali sadrži sve ostale.
Takođe može da se navede i treći argument prilikom operacije isecanja, što predstavlja vrednost *koraka* pri sečenju (po podrazumevanom podešavanju, korak je veličine 1):

```
>>> listakupovine = ['jabuka', 'mango', 'sargarepa', 'banana']
>>> listakupovine[::1]
['jabuka', 'mango', 'sargarepa', 'banana']
>>> listakupovine[::2]
['jabuka', 'sargarepa']
>>> listakupovine[::3]
['jabuka', 'banana']
>>> listakupovine[::-1]
['banana', 'sargarepa', 'mango', 'jabuka']
```

Primetite da kada je vrednost koraka 2, dobijamo stavke sa pozicija 0, 2, ... Kada je korak veličine 3, dobijamo stavke sa pozicija 0, 3, itd.
Probajte različite kombinacije tih specifikacija operacije sečenja. Koriste Python interpreter tj prompt, tako da momentalno možete da vidite rezultate. Najbolja stvar u sekvencama je ta, da možete pristupiti tupleima, listama i stringovima na identičan način!

11.5 Set (skup)

Setovi (skupovi) su nesređene zbirke jednostavnih objekata. Oni se koriste ukoliko je postojanje objekata u kolekciji važnije od reda u kom se nalaze ili od toga koliko puta se objekat pojavljuje u kolekciji.
Korišćenjem setova, možete vršiti test članstva, da li je podskup drugog seta, naći zajedničke elemente između dva seta, i tako dalje.

```
>>> bri = set(['brazil', 'rusija', 'indija'])
```

```
>>> 'indija' in bri
True
>>> 'usa' in bri
False
>>> brik = bri.copy()
>>> brik.add('kina')
>>> brik.issuperset(bri)
True
>>> bri.remove('rusija')
>>> bri & brik  # ili bri.intersection(brik)
{'brazil', 'indija'}
```

Kako to funkcioniše:
Primer je prilično samoobjašnjiv jer obuhvata osnovne teorije o skupovima iz matematike koju ste učili u školi.

11.6 Reference

Kada napravite neki objekat i dodelite ga nekoj promenljivoj, promenljiva ima *neki odnos* (referencu) prema tom objektu i ne predstavlja sam taj objekat! To jest, ime promenljive ukazuje na deo memorije računara gde je uskladišten taj objekat. To se zove izgradnja (definisanje-**binding**) imena objekta. Možete da mislite o tome kao o prečicama na glavnom ekranu računara - možete da napravite prečicu, izbrišete, kopirate, ali program na koji se odnosi prečica i dalje ostaje. Ali postoje i izuzeci!
Generalno, ne treba da se brinete o tome, ali postoji nuspojave koje nastaju prilikom upotrebe referenci za koje treba da budete obazrivi:
Primer (sačuvati kao `reference.py`):

```
print('Jednostavno dodavanje vrednosti')
listakupovine = ['jabuka', 'mango', 'sargarepa', 'banana']
mojalista = listakupovine  # mojalista je samo drugo ime koje se odnosi
# na isti objekat (listakupovine)!

del listakupovine[0]  # Kupili smo prvu stvar, pa je brisemo iz liste
print('listakupovine je', listakupovine)
print('mojalista je', mojalista)
# obratite paznju da obe (i listakupovine i mojalista) prikazuju isti
# rezultat
# bez objekta 'jabuka', cime potvrdjujemo da one referisu na isti
# objekat

print('Kopiranje pomocu punog isecanja')
mojalista = listakupovine[:]  # pravimo kopiju tako sto isecamo sve
# objekte iz jedne liste

del mojalista[0]  # uklanjamo prvi objekat
print('listakupovine je', listakupovine)
print('mojalista je', mojalista)
```

```
# sada primetite da su liste razlicite!
```

Izlaz:

```
$ python3 reference.py
Jednostavno dodavanje vrednosti
listakupovine je ['mango', 'sargarepa', 'banana']
mojalista je ['mango', 'sargarepa', 'banana']
Kopiranje pomocu punog isecanja
listakupovine je ['mango', 'sargarepa', 'banana']
mojalista je ['sargarepa', 'banana']
```

Kako to funkcioniše:
Većina stvari je objašnjenja u komentarima unutar programa.
Zapamtite da ako želite da napravite kopiju liste ili neke slične vrste sekvenci ili kompleksnih objekata (ne mislimo na jednostavne *objekte* kao što su npr celi brojevi), onda morate da koristite operaciju isecanja kako bi napravi kopiju originalnog objekta. Ako samo dodelite drugo ime varijable, oba imena će se "odnositi" na isti objekat i to bi moglo da bude problem ako niste dovoljno pažljivi.
Napomena za Perl programere
Zapamtite da naredba dodele imena za liste **ne** kreira kopiju iste. Morate da koristite operaciju odsecanja da napravite kopiju liste.

11.7 Više o stringovima

Već smo već razmatrali stringove u pojedinostima ranije. Šta više bi trebali da znate? Pa, da li ste znali da su stringovi takođe objekti i imaju metode koji mogu da rade sve, od provere delova stringa pa da menjanja i uklanjanja razmaka između reči unutar stringa?
Stringovi koje koristite u programima su objekti koji pripadaju klasi `str` . Neke korisne metode ove klase su pokazane u sledećem primeru. Za kompletnu listu takvih metoda, pogledajte `help(str)` .
Primer (sačuvati kao `str_metode.py`):

```
ime = 'Swaroop'  # ovo je prvi string objekt

if ime.startswith('Swa'):
    print('Da, string pocinje sa "Swa"')

if 'a' in ime:
    print('Da, string u sebi ima slovo "a"')

if ime.find('war') != -1:
    print('Da, string sadrzi slova "war"')

razmak = '_*_'
mojalista = ['brazil', 'Rusija', 'Indija', 'Kina']
print(razmak.join(mojalista))
```

Izlaz:

```
$ python3 str_metode.py
```

```
Da, string pocinje sa "Swa"
Da, string u sebi ima slovo "a"
Da, string sadrzi slova "war"
brazil_*_Rusija_*_Indija_*_Kina
```

Kako to funkcioniše:
Ovde vidimo dosta metoda stringova u akciji. `startswith` metod se koristi da sazna da li string počinje sa datim stringom. `in` operator koristimo za proveru da li je dati string deo proveravanog stringa.
`find` metod se koristi za lociranje položaja datog substringa unutar stringa, `find` vraća -1 ako je neuspešan u pronalaženju substringa. `str` klasa ima koristan metod da `join` (spoji) stavke iz niza sa stringom kojim mi to želimo, koji služi kao razdvajanje između svake stavke u redosledu i vraća nam veći niz generisan od toga.

11.8 Rezime

Ispitivali smo detaljno, različite strukture podataka ugrađene u Python. Ove strukture podataka će biti od suštinskog značaja za pisanje programa neke razumne veličine.
Sada kada imamo puno osnovnih znanja o Python-u, sledeće što ćemo razmatrati je kako da osmislimo i napišemo neki koristan Python program.

12 Rešavanje problema

Ispitivali smo različite delove Python jezika, a sada ćemo da pogledamo kako da sve ove delove uklopimo u projektovanje i pisanje programa koji *rade* nešto korisno. Ideja je da naučite kako da samostalno pišete skripte u Python-u.

12.1 Problem

Problem koji želimo da rešimo je "*Ja želim program koji kreira rezervnu kopiju svih mojih važnih datoteka*".
Iako je ovo prilično jednostavan problem, ne postoji dovoljno informacija za nas da počnemo sa njegovim rešavanjem. Neophodno je izvršiti malu **analizu**. Na primer, kako da odreditmo *koje* datoteke treba da budu obuhvaćene? *Kako* će se one čuvati? *Gde* će se one čuvati?
Nakon dobre analize problema, mi možemo da počnemo da **dizajniramo** naš program. Mi pravimo spisak stvari o tome kako naš program treba da radi. U ovom slučaju, JA sam napravio sledeću listu, kako bih *JA* to želeo da izvedem. Ako Vi uradite dizajn, ne morate doći do takvog rešenja, jer svako ima neki svoj način obavljanja nekog posla i rešavanja problema, tako da je različitost rešavanja sasvim u redu.

- Fajlovi i direktorijumi koji će da budu bekapovani su navedeni u listi.
- Bekap se mora čuvati u glavnom bekap direktorijumu.
- Fajlovi se bekapuju u `zip` fajl.
- Naziv `zip` arhive je trenutni datum i vreme.
- Mi koristimo standardnu `zip` komandu koja je podrazumevano dostupna u bilo kojoj standardnooj Linux/Unix distribuciji. Korisnici Windows-a mogu da je instaliraju sa internet strane GnuWin32 projekta i dodati `C:\Program Files\GnuWin32\bin` u sistemsku promenljivu okruženja PATH, slično kao što smo uradili za prepoznavanje komandi u Python-u . Imajte na umu da možete da koristite bilo koji program za arhiviranje ako želite, i ako znate komande u njemu, dokle god se u tom programu može raditi i iz komandne linije.

12.2 Rešenje

Kako je dizajn našeg programa sad prilično stabilan, možemo početi pisati kod koji je **realizacija** našeg rešenja.
Sačuvaj kao `backup_ver1.py` :

```
import os
import time

# 1. Fajlovi i folderi koje zelimo da back-up-ujemo se specifirani u
# listi
izvor = ['C:\\py', '"C:\Documents and Settings\Pure\Desktop\V0.0\"']
# Primetite da smo koristili duple navodnike unutar stringa, zbog imena
# koja sadrze razmake

# 2. Back-up ce biti sacuvan u glavnom back-up direktorijumu
ciljni_dir = 'D:\\Resto' # Zapamtite da promenite ovo, unesite lokaciju
# koja vama odgovara na vasem racunaru
```

```
# 3. Fajlovi se back-up-uju u zip fajl
# 4. Ime zip arhive je trenutni datum i vreme
cilj = ciljni_dir + os.sep + time.strftime('%Y%m%d%H%M%S') + '.zip'

# 5. Koristimo zip komandu da posaljemo fajlove u zip arhivu
zip_komanda = "zip -qr {0} {1}".format(cilj, ' '.join(izvor))

# Pokrece back-up
if os.system(zip_komanda) == 0:
    print('Uspesno smo izvrsili back-up u', cilj)
else:
    print('BACK-UP NIJE USPEO!')
```

Izlaz:

```
$ python backup_ver1.py
Uspesno smo izvrsili back-up u D:\Resto\20130407215156.zip
```

Sada smo u fazi **testiranja**, gde smo proveravali da naš program radi ispravno. Ako se program ne ponaša kako očekujemo, onda moramo da izvršimo **debagovanje** programa, odnosno da uklonimo *greške* (bagove) u programu.

Ako Vam gornji program ne radi, upišite `print(zip_komanda)` neposredno pre poziva komande `os.system` a zatim i pokrenite program. Sada uradite copy/paste izlaza `zip_komanda` u terminal Vašeg operativnog sistema i proverite da li ispravno radi sama. Ako ta komanda ne uspe, proverite uputstvo za zip komandu o tome šta bi mogao da bude problem. Ako ova komanda "uradi posao", onda proverite svoj Python program, jer mora da se tačno podudara programu navedenom iznad. (takođe proverite da li direktorijum u koji vršite bekap postoji - ovaj program NE KREIRA taj direktorijum za Vas).

Kako to funkcioniše:

Primetite kako smo pretvarali naš *dizajn* u *program* - korak-po-korak.

Mi omogućavamo korišćenje `os` i `time` modula tako što ćemo ih prvo importovati. Zatim smo naveli fajlove i direktorijume za koje pravimo rezervne kopije u listi `izvor`. Ciljni direktorijum je mesto gde čuvamo ove rezervne kopija i to je navedeno u `ciljni_dir` varijabli. Naziv zip arhive koju ćemo stvoriti je trenutni datum i vreme koje smo generisali pomoću `time.strftime()` komande. Takođe će imati `.zip` ekstenziju i čuvaće se u `ciljni_dir` direktorijumu.

Obratite pažnju na korišćenje `os.sep` promenljive - to daje separator direktorijuma prema vašem operativnom sistemu, odnosno biće '`/`' na Linux-u i Unix-u, '`\\`' na Windows-u i '`:`' na MacOS-u. Korišćenje `os.sep` umesto ovih znakova direktno će učiniti naš program prenosiv (portabl) i moći će da radi na svim ovim sistemima.

`time.strftime()` funkcija preuzima argumente, slične onaima koje smo koristili u navedenom programu. `%Y` oznaka će biti zamenjena godinom. `%m` specifikacija će biti zamenjena sa mesecom predstavljenim kao decimalni broj između 01 i 12 i tako dalje. Kompletan spisak takvih specifikacija može se naći u Reference i uputstava Python-a.

Mi kreiramo ime ciljne zip datoteke koristeći operator sabiranja koji *koncetriše* stringove odnosno spaja dva stringa zajedno i vraća nov. Zatim ćemo napraviti string `zip_komanda` koji sadrži komandu koju želimo izvršiti. Možete da proverite da li ova komanda radi tako što ćete je pokrenuti u šelu (Linux terminal ili DOS prompt).

`zip` naredba koja se koriste ima neke opcije i parametre koje smo joj dali. `-q` se koristi da označi da

zip komanda radi tiho (ne pravi izlaz u terminalu). `-r` određuje da komanda radi rekursivno za direktorijume tj treba da sadrži sve poddirektorijume i datoteke. Dve opcije se kombinuju i skraćeno pišu kako `-qr` . Opcije koje su dalje navedene su ime zip arhive i potom sledi spisak fajlova i direktorijuma za bekap. Mi konvertujemo listu `izvor` u string koristeći `join` metod stringova koji smo već videli kako se koristi.
Onda smo konačno *pokrenuli* komandu koristeći `os.system` funkciju koja pokreće komandu kao da je pokrenuti iz *sistema*, odnosno u šelu - ona vraća 0 ako je komanda uspešno izvršena, u suprotnom vraća broj greške.
U zavisnosti od ishoda komande, mi smo prikazali odgovarajuću poruku da li je bekap uspeo ili ne.
To je to, mi smo napravili skriptu koja pravi rezervnu kopiju naših važnih fajlova!
Napomena za korisnike operativnog sistema Windows
Umesto duplih obrnutih kosih crta (\\), takođe možete koristiti raw string. Na primer, koristite `'C:\\Documents'` ili `r'C:\Documents'` . Međutim, ne koristite `'C:\Documents'` jer bi to Vaš OS protumačio kao nepoznatu sekvencu `\D` .
Sada kada imamo skriptu koja radi rezervne kopije, možemo je koristiti kad god želimo da imamo rezervne kopije datoteka. Linux/Unix korisnicima se savetuje da koriste metodu za izvršavanje programa kao što je opisano ranije, tako da mogu pokrenuti bekap skriptu bilo kada i bilo gde. Ovo se zove **operativna** faza ili dostavna (**deployment**) faza u razvoju softvera.
Gornji program radi ispravno, ali (kao i obično) prvi programi ne rade uvek onako kako ste očekivali. Na primer, možda postoji problem, ako niste osmislili program pravilno, ili ako ste napravili grešku pri kucanju koda, itd. Shodno tome, s vremena na vreme, moraćete da se vraćate u fazu projektovanja ili ćete morati da debagujete svoj program.

12.3 Druga verzija

Prva verzija naše skripte radi. Međutim, možemo napraviti neke male izmene na njoj, kako bi mogla da radi bolje u svakodnevnoj upotrebi. Ovo se zove faza **održavanja** softvera.
Jedna od izmena koja bi mi koristila je bolji mehanizam imenovanje datoteka - koristeći trenutno *vreme* kao ime fajla unutar direktorijuma sa imenom koji je trenutni *datum*, koji je direktorijum u okviru glavnog bekap direktorijuma. Prva prednost je u tome što bi se vaše kopije čuvale sortirane na hijerarhijski način i zato bi nam bilo mnogo lakše da se snađemo u njima. Druga prednost je u tome što su imena fajlova mnogo kraća. Treća prednost odvajanja direktorijuma, je da će nam pomoći da proverite da li ste napravili rezervnu kopiju za taj dan, jer će direktorijum (koji ima naziv koji je trenutni datum) biti kreiran samo ako ste napravili rezervnu kopiju za taj dan.
Sačuvaj kao `backup_ver2.py` :

```
import os
import time

# 1. Fajlovi i folderi koje zelimo da back-up-ujemo se specifirani u
# listi
izvor = ['C:\\py', '"C:\Documents and Settings\xinjure\Desktop\V0.0\"']
# Primetite da smo koristili duple navodnike unutar stringa, zbog imena
# koje sadrze razmake

# 2. Back-up ce biti sacuvan u glavnom back-up direktorijumu
ciljni_dir = 'D:\\Resto' # Zapamtite da promenite ovo, unesite lokaciju
# koja vama odgovara na vasem racunaru
```

```
# 3. Fajlovi se back-up-uju u zip fajl
# 4. Trenutni datum je ime poddirektorijuma u glavnom direktorijumu
danas = ciljni_dir + os.sep + time.strftime('%Y%m%d')
# Trenutno vreme je ime zip arhive
sada = time.strftime('%H%M%S')

# Kreiranje poddirektorijuma ukoliko on ne postoji
if not os.path.exists(danas):
    os.mkdir(danas)  # Kreira direktorijum
    print('Uspesno smo kreirali direktorijum', danas)

# Ime zip arhive
cilj = danas + os.sep + sada + '.zip'

# 5. Koristimo zip komandu da posaljemo fajlove u zip arhivu
zip_komanda = "zip -qr {0} {1}".format(cilj, ' '.join(izvor))

# Pokrece back-up
if os.system(zip_komanda) == 0:
    print('Uspesno smo izvrsili back-up u', cilj)
else:
    print('BACK-UP NIJE USPEO!')
```

Izlaz:

```
$ python3 backup_ver2.py
Uspesno smo kreirali direktorijum D:\Resto\20130408
Uspesno smo izvrsili back-up u D:\Resto\20130408\131658.zip
```

Kako to funkcioniše:
Veći deo našeg programa ostaje isti. Jedine promene su to, da smo proverili da li postoji direktorijum sa imenom današnjeg dana unutar glavnog bekap direktorijuma pomoću funkcije `os.path.exists`. Ako ne postoji, mi ga kreiramo pomoću `os.mkdir` funkcije.

12.4 Treća verzija

Druga verzija radi prilično dobro kad radima bekap dosta fajlova, ali kada postoji mnogo rezervnih kopija, teško je razlikovati gde su/od čega su bekap-ovi! Na primer, možda sam napravio neke velike promene u programu ili prezentaciji, pa zato želim da mi ime zip arhive govori koje su to promene/verzije programa. To se može lako postići dodavanjem korisnikovog komentara u ime zip arhive.

Pažnja!
Sledeći program ne radi, tako da ne brinite, uradite ga, jer ćete izvući jednu pouku iz primera.
Sačuvaj kao `backup_ver3.py` :

```
import os
import time
```

```
# 1. Fajlovi i folderi koje zelimo da back-up-ujemo se specifirani u
# listi
izvor = ['C:\\py', '"C:\Documents and Settings\xinjure\Desktop\V0.0\"']
# Primetite da smo koristili duple navodnike unutar stringa, zbog imena
# koje sadrze razmake

# 2. Back-up ce biti sacuvan u glavnom back-up direktorijumu
ciljni_dir = 'D:\\Resto' # Zapamtite da promenite ovo, unesite lokaciju
# koja vama odgovara na vasem racunaru

# 3. Fajlovi se back-up-uju u zip fajl
# 4. Trenutni datum je ime poddirektorijuma u glavnom direktorijumu
danas = ciljni_dir + os.sep + time.strftime('%Y%m%d')
# Trenutno vreme je ime zip arhive
sada = time.strftime('%H%M%S')

# Trazi od korisnika da unese komentar, koji bi se primenio u imenu zip
# arhive
komentar = input('Prilozite komentar back-up-a --> ')
if len(komentar) == 0:  # proverava da li je komentar unet
    cilj = danas + os.sep + sada + '.zip'
else:
    cilj = danas + os.sep + sada + '_' +
    komentar.replace(' ', '_') + '.zip'

# Kreiranje poddirektorijuma ukoliko on ne postoji
if not os.path.exists(danas):
    os.mkdir(danas)  # Kreira direktorijum
    print('Uspesno smo kreirali direktorijum', danas)

# 5. Koristimo zip komandu da posaljemo fajlove u zip arhivu
zip_komanda = "zip -qr {0} {1}".format(cilj, ' '.join(izvor))

# Pokrece back-up
if os.system(zip_komanda) == 0:
    print('Uspesno smo izvrsili back-up u', cilj)
else:
    print('BACK-UP NIJE USPEO!')
```

Izlaz:

```
$ python3 backup_ver3.py
  File "backup_ver3.py", line 22
    cilj = danas + os.sep + sada + '_' +
                                       ^
SyntaxError: invalid syntax
```

Kako ovo (ne)radi:

Ovaj program ne radi! Python nam govori da postoji greška u sintaksi, što znači da skripta nema strukturu koju Python očekuje. Kada pažljivo osmotrimo grešku koju nam je izbacio Python, takođe vidimo da nam je pokazao mesto gde je otkrio grešku. Tako da možemo da započnemo da *debagujemo* naš program, tj da počnemo da ispravljamo grešaka od te linije pa nadalje (ako postoje još neke).
Pažljivim posmatranjem našeg programa, vidimo da je jedna logička linija podeljena na dve fizičke linije, a mi nismo precizirali da ove dve fizičke linije idu zajedno. U suštini, Python je našao operator dodavanja (+) bez operanda u tom logičkom redu, pa zbog toga ne zna kako da nastavi izvršavanje programa. Zapamtite da možemo navesti da je nastavak logičke linije u sledećoj fizičkoj liniji upotrebom kose crte na kraju fizičke linije. Dakle, fiksirajmo ovu grešku unutar našeg programa. Ove korekcije programa, kada naiđemo na greške se naziva **fiksiranje bagova**.

12.5 Četvrta verzija

Sačuvaj kao `backup_ver4.py` :

```
import os
import time

# 1. Fajlovi i folderi koje zelimo da back-up-ujemo se specifirani u
# listi
izvor = ['C:\\py', '"C:\Documents and Settings\xinjure\Desktop\V0.0\"']
# Primetite da smo koristili duple navodnike unutar stringa, zbog imena
# koje sadrze razmake

# 2. Back-up ce biti sacuvan u glavnom back-up direktorijumu
ciljni_dir = 'D:\\Resto' # Zapamtite da promenite ovo, unesite lokaciju
# koja vama odgovara na vasem racunaru

# 3. Fajlovi se back-up-uju u zip fajl
# 4. Trenutni datum je ime poddirektorijuma u glavnom direktorijumu
danas = ciljni_dir + os.sep + time.strftime('%Y%m%d')
# Trenutno vreme je ime zip arhive
sada = time.strftime('%H%M%S')

# Trazi od korisnika da unese komentar, koji bi se primenio u imenu zip
# arhive
komentar = input('Prilozite komentar back-up-a --> ')
if len(komentar) == 0:  # proverava da li je komentar unet
    cilj = danas + os.sep + sada + '.zip'
else:
    cilj = danas + os.sep + sada + '_' + \
    komentar.replace(' ', '_') + '.zip'

# Kreiranje poddirektorijuma ukoliko on ne postoji
if not os.path.exists(danas):
    os.mkdir(danas)  # Kreira direktorijum
    print('Uspesno smo kreirali direktorijum', danas)
```

```
# 5. Koristimo zip komandu da posaljemo fajlove u zip arhivu
zip_komanda = "zip -qr {0} {1}".format(cilj, ' '.join(izvor))

# Pokrece back-up
if os.system(zip_komanda) == 0:
    print('Uspesno smo izvrsili back-up u', cilj)
else:
    print('BACK-UP NIJE USPEO!')
```

Izlaz:

```
$ python3 backup_ver4.py
Prilozite komentar back-up-a --> dodat novi primer
Uspesno smo izvrsili back-up u
D:\Resto\20130408\150329_dodat_novi_primer.zip
$ python3 backup_ver4.py
Prilozite komentar back-up-a -->
Uspesno smo izvrsili back-up u D:\Resto\20130408\150605.zip
```

Kako to funkcioniše:

Ovaj program sada radi! Prođimo kroz stvarna poboljšanja koje smo napravili u verziji 3. Mi tražimo od korisnika da unese neki komentar bekapa koristeći `input` funkciju, a zatim proverimo da li je korisnik zaista uneo nešto, tako što saznamo dužinu unosa pomoću `len` funkcije. Ukoliko korisnik pritisne samo taster `enter` bez ikakvog unosa (možda je to bio samo uobičajeni bekap ili nisu napravljene nikakve posebne promene), onda program radi na isti način kao što je radio ranije.

Međutim, ako je korisnik uneo neki komentar, onda taj komentar dodajemo u ime zip arhive neposredno pre `.zip` ekstenzije. Primetite da smo zamenili razmake u komentaru sa donjom crticom - to radimo zato što je korišćenje fajlova bez razmaka u imenima mnogo lakše.

12.6 Više prilagođavanja programa

Četvrta verzija je sasvim zadovoljavajuća skripta za rad većine korisnika, ali uvek postoji prostor za poboljšanja. Na primer, možete uključiti sintaksu programa gde možete da izaberete komandu `-v` da ovaj program počne da prikazuje poruke na ekranu (*verbosity*).

Drugo moguće unapređenje bi bilo da omogućite dodavanje fajlova i direktorijuma koje želimo da bekapujemo, tako što bi ih prosleđivali skripti preko komandne linije. Možemo da saznamo ova imena iz `sys.argv` liste, pa ih možemo da dodamo u našu `izvor` listu pomoću `extend` metode, koja je metoda `list` klase.

Najvažnije unapređenje bi bilo da ne koristite `os.system` za kreiranje arhive, već korišćenjem `zipfile` ili `tarfile` ugrađenih modula za kreiranje ovih arhiva. Ovi moduli su deo standardne biblioteke i dostupni su za vas da ih koristite bez spoljnih zavisnosti prema zip programu koji mora biti instaliran na vašem računaru.

Međutim, ja sam koristio `os.system` način stvaranja bekapa u gore navedenim primerima čisto iz pedagoških razloga, jer je primer dovoljno jednostavan da se razume, i "radi" solidno da bi bio i koristan.

Možete li pokušati da napišete petu verziju programa koja koristi zipfile modul umesto pozivanja `os.system` -a?

12.7 Proces razvoja softvera

Do sada smo prošli kroz razne **faze** u procesu pisanja softvera. Ove faze se mogu sažeti na sledeći način:

1. Šta (Analiza)
2. Kako (dizajn)
3. Napraviti (implementacija-realizacija)
4. Test (testiranje i otklanjanje grešaka)
5. Korišćenje (Operacionalizacija ili Deployment)
6. Održavanje (Unapređivanje)

Preporučen način pisanja programa je postupak kroz koji smo prolazili u kreiranju bekap skripte: Uraditi analizu i dizajn. Počnite sa implementacijom jednostavne verzije. Testiranje i otklanjanje grešaka. Korišćenje da se osigura da sve radi kao što je i očekivano. Tada dodajete neke nove funkcije koje želite a zatim se ponavlja Napravi - Testiraj - Koristi - Održavaj ciklus onoliko puta koliko je potrebno (za kvalitetne programe beskonačno dugo). Zapamtite, **softver se uzgaja, a ne izgrađuje**.

12.8 Rezime

Videli smo kako da kreirate sopstvene programe/Python skripte kroz razne faze koji su uključene u pisanje takvih programa. Možda će vam sve to biti od koristi prilikom stvaralja nekog Vašeg sopstvenog programa, kao što smo mi činili u ovom poglavlju, tako da vam postane konforan rad u Python-u, kao i u rešavanju problema.
U sledećem poglavlju ćemo razmatrati objektno orijentisano programiranje.

13 Objektno orijentisano programiranje

U svim programima koje smo pisali do sada, dizajnirali smo naše programe sa funkcijama, odnosno blokovima komandi koje obrađuju podatke. To se zove *proceduralno orijentisan* način programiranja. Postoji još jedan način organizovanja programa a to je da kombinujemo podatke i funkcionalnosti i od svega toga napravimo nešto što se zove objekat. To se zove *Objektno orijentisano programiranje*. U većinu slučajeva možete koristiti proceduralni način programiranja, ali prilikom pisanja velikih programa ili rešavanju nekih problema je bolje koristiti objektno orijentisanu tehniku programiranja.

Klase i objekti su dva glavna aspekta objektno orijentisanog programiranja. **Klasa** stvara novu *vrstu* (tip), gde je **objekat** *instanca* klase. Analogija je da možete da imate promenljive tipa `int` što znači, u prevodu, da varijabla koja čuva celobrojnu vrednost je, u stvari instanca (objekat) unutar `int` klase.

Napomena za programere statičkih jezika

Imajte na umu da se čak i celi brojevi (integeri) tretiraju kao objekti (od `int` klase). Ovo se dosta razlikuje od C++ i Jave (pre verzije 1.5), gde su celi brojevi primitivni nativni tipovi. Pogledajte `help(int)` za više detalja o klasi.

C# i Java 1.5 programeri će videti ovo kao veoma sličan konceptu *boxing* i *unboxing*.

Objekti mogu da skladište podatke koristeći obične promenljive koje *pripadaju* tom objektu. Promenljive koje pripadaju objektu ili klasi se nazivaju **polja** (fields). Objekti takođe mogu da imaju funkcionalnosti, uz pomoć funkcija koje *pripadaju* klasi. Takve funkcije se nazivaju **metode** klase. Ova terminologija je važno jer nam pomaže da se napravimo razliku između funkcija i promenljivih koje su nezavisne i onih koji pripadaju klasi ili objektu. Kolektivno, polja i metode nama mogu biti označeni kao **atributi** te klase.

Polja mogu biti dve vrste - oni mogu da pripadaju svakoj instanci/objektu neke klase ili mogu da pripadaju samoj klasi. Zato se, navedenim redom nazivaju **promenljive instance** i **promenljive klase**.

Klasa se kreira pomoću ključne reči `class`. Polja i metode te klase su navedeni u uvučenim blokovima.

13.1 self

Metode klase imaju samo jednu specifičnu razliku od običnih funkcija - oni moraju da imaju dodatno prvo ime koje mora da se doda na početak liste parametara, ali joj mi **ne dodeljujemo** nikakvu vrednosti (za ovaj parametar) kada pozivamo metod iz našeg programa, već će mu vrednost obezbediti sam Python. Ova posebna promenljiva odnosi se na *sam taj* objekat, i po običaju, daje joj se ime `self` .

Mada možete dati bilo koje ime za ovaj parametar, *preporučuje se* da koristite ime `self` - svako drugo ime je definitivno neodobreno. Postoje mnoge prednosti korišćenja standardnog imena - svaki čitalac vašeg programa će ga odmah prepoznati, pa čak i specijalizovani IDE (Integrated Development Environments) može da vam pomogne ako koristite `self` .

Napomena za C++/Java/C# programere

`self` u Python-u je ekvivalentno `this` pokazivaču u C++ i `this` referenci u Javi i C#.

Sigurno se pitate kako Python daje vrednost za `self` i zašto mi ne treba da dodeljujemo vrednost za nju. Primer će sve ovo da razjasni. Recimo da imate klasu `MojaKlasa` i instancu ove klase koja se zove `mojobjekt` . Kada pozovete metod ovog objekta kao `mojobjekt.metod(arg1, arg2)` , ovo će ce u Python-u automatski konvertovati u `MojaKlasa.metod(mojobjekt, arg1, arg2)`, i to je ono šta je specijalno u `self` -u.

To takođe znači da ako imate metod koji ne uzima nikakve argumente, onda ćete još uvek imati jedan argument - `self` .

13.2 Klase

Najjednostavnija moguća klasa je prikazana u sledećem primeru (sačuvajte kao `najjednostavnijaklasa.py`).

```
class Licnost:
    pass  # prazan blok

L = Licnost()
print(L)
```

Izlaz:

```
$ python3 najjednostavnijaklasa.py
<__main__.Licnost object at 0x00CA21D0>
```

Kako to funkcioniše:
Mi stvaramo novu klasu koristeći `class` naredbu i ime klase. Ovo je praćeno uvučenim blokom naredbi koje čine telo klase. U ovom slučaju, imamo prazan blok koji smo definisali pomoću `pass` naredbe.
Sledeće što smo uradili je da smo kreirali objekat/instancu ove klase koristeći ime klase praćeno zagradama (Mi ćemo naučiti više o ovome načinu u sledećem odeljku). Za proveru rada, mi smo potvrdili tip promenljive, jednostavno ga prikazujući na ekranu. To nam govori da imamo instancu od `Licnost` klase u `__main__` modulu.
Primetite da je adresa memorije računara, u kojoj se čuva vaš objekat, je takođe prikazana. Adresa će imati neku drugu vrednost na Vašem računaru, jer Python može da skladišti objekat gde nađe prostora za njega.

13.3 Metode objekata

Već smo objasnili da klase/objekti mogu da imaju metode, baš kao i funkcije, osim što imamo ekstra `self` promenljivu. Sada ćemo videti primer (sačuvajte kao `metod.py`).

```
class Licnost:
    def kaziZdravo(self):
        print('Zdravo, kako si?')

L = Licnost()
L.kaziZdravo()

# Ovaj kratak primer moze biti napisan i kao Licnost().kaziZdravo()
```

Izlaz:

```
$ python3 metod.py
Zdravo, kako si?
```

Kako to funkcioniše:
Ovde vidimo `self` u akciji. Primetimo da `kaziZdravo` metoda "neprima" parametre, ali i dalje

ima `self` u definiciji funkcije.

13.4 Init metod

Postoji mnogo različitih imena metoda koja imaju poseban značaj Python-ovim klasama. Mi ćemo sada videti značaj `__init__` metode.
`__init__` metod se pokreće u trenutku kada se objekat klase inicijalizuje. Metod je koristan za obavljanje bilo kakve *inicijalizacije* koju želite da uradite sa svojim objektom. Primetite duplu donju crticu i na početku i na kraju imena metoda.
Primer (sačuvati kao `klasa_init.py`):

```
class Licnost:
    def __init__(self, ime):
        self.ime = ime
    def kaziZdravo(self):
        print('Zdravo, ja se zovem', self.ime)

L = Licnost('Swaroop')
L.kaziZdravo()

# Ovaj kratki primer se takodje moze napisati kao
Licnost('Swaroop').kaziZdravo()
```

Izlaz:

```
$ python3 klasa_init.py
Zdravo, ja se zovem Swaroop
```

Kako to funkcioniše:
Ovde smo definisali `__init__` metod koji kao parametar uzima `ime` (zajedno sa uobičajenim `self`). Zatim smo kreirali novo polje koje se takođe zove `ime`. Obratite pažnju da su to dve različite promenljive iako su obe nazvane '`ime`'. Tu nema nikakvih problema, jer oznaka sa tačkom `self.ime` znači da postoji nešto što se zove "`ime`" i da je to nešto deo objekta pod nazivom "`self`" a drugo `ime` je lokalna promenljiva. Pošto smo eksplicitno ukazali na koje ime smo mislili, ne postoji konfuzija.
Najvažnije je primetiti da mi nismo direktno pozvali `__init__` metod, već smo postavili argumente unutar zagrada nakon imena klase, prilikom kreiranja nove instance klase. Ovo je poseban značaj ove metode.
Sada smo u mogućnosti da koristimo polje `self.ime` u našim metodama, na primeru se prikazalo u `kaziZdravo` metodu.

13.5 Promenljive klasa i objekata

Već smo razgovarali o funkcionalnim delovima klasa i objekata (tj. metodama), sada ćemo učiti o njihovom delu podataka. Deo sa podacima, odnosno polja, nije ništa drugo nego obične varijable koje su *vezane* sa specifikacijama imena (**namespaces**) za klase i objekte. To znači da ova imena važe samo unutar ovih klasa i objekata. Zato se zovu *name spaces*.
Postoje dve vrste *polja* -promenljive klase i promenljive objekata koji su klasifikovani u zavisnosti od toga da li pripadaju klasi ili objektu, odnosno da li klasa ili objekat *poseduje* tu varijablu.
Promenljive klase su deljive - mogu biti dostupne svim instancama te klase. Postoji samo jedan

primerak promenljive klase i kada neki objekat napravi promenu na promenljivoj klase, ta promena će biti vidljiva svim drugim objektima te klase.
Promenljive objekta su u vlasništvu svakog **individualnog** objekta/instance klase. U ovom slučaju, svaki objekat ima svoj primerak polja, odnosno one ne dele i ne odnose se na bilo koji način prema polju sa istim imenom u nekoj drugoj instanci. Primer će ovo lako objasniti (sačuvajte kao `objvar.py`):

```
class Robot:
    '''Predstavlja robota sa imenom.'''

    # varijabla klase, broji kolicinu robota
    populacija = 0

    def __init__(self, ime):
        '''Inicijalizacija podataka'''
        self.ime = ime
        print('(Inicijalizujem {0})'.format(self.ime))

        # kada je kreirana licnost robota
        # povecava se populacija
        Robot.populacija += 1

    def __del__(self):
        '''Umirem.'''
        print('{0} je unisten!'.format(self.ime))

        Robot.populacija -= 1

        if Robot.populacija == 0:
            print('{0} je bio poslednji.'.format(self.ime))
        else:
            print('Funkcionalno je jos {0:d}
robota'.format(Robot.populacija))

    def kaziZdravo(self):
        '''Robot vas pozdravlja.

        Da, oni mogu i to!'''
        print('Pozdravljam Vas, moj Gospodaru, mozete me zvati
{0} .'.format(self.ime))

    def koliko():
        '''Prikazuje trenutnu populaciju.'''
        print('Imamo {0:d} robota'.format(Robot.populacija))
    koliko = staticmethod(koliko)

droid1 = Robot('R2-D2')
```

```
droid1.kaziZdravo()
Robot.koliko()

droid2 = Robot('C-3P0')
droid2.kaziZdravo()
Robot.koliko()

print("\nRoboti mogu da rade nesto ovde.\n")

print("Roboti su zavrsili svoj rad. Hajde da ih unistimo.")
del droid1
del droid2

Robot.koliko()
```

Izlaz:

```
$ python3 objvar.py
(Inicijalizujem R2-D2)
Pozdravljam Vas, moj Gospodaru, mozete me zvati R2-D2 .
Imamo 1 robota
(Inicijalizujem C-3P0)
Pozdravljam Vas, moj Gospodaru, mozete me zvati C-3P0 .
Imamo 2 robota

Roboti mogu da rade nesto ovde.

Roboti su zavrsili svoj rad. Hajde da ih unistimo.
R2-D2 je unisten!
Funkcionalno je jos 1 robota
C-3P0 je unisten!
C-3P0 je bio poslednji.
Imamo 0 robota
```

Kako to funkcioniše:

Ovo je dugačak primer ali pomaže da pokažemo prirodu varijabli klase i objekata. Ovde, `populacija` pripada `Robot` klasi i stoga je to promenljiva klase. Promenljiva `ime` pripada objektu (dodelimo je objektu koristeći `self`) i stoga je promenljiva objekta.

Zbog toga, mi smo promenljivu klase `populacija` pozivali kao `Robot.populacija` a ne kao `self.populacija`. Mi pozivamo promenljivu objekta `ime` koristeći `self.ime` oznaku unutar metoda tog objekta. Zapamtite ovu jednostavnu razliku između promenljivih klasa i objekata. Takođe, imajte na umu da će promenljiva objekta sa istim imenom kao promenljiva klase, da sakrije promenljivu klase!

`koliko` je zapravo metoda koja pripada klasi, a ne objektu. To znači da možemo da je definišemo kao `classmethod` ili `staticmethod`, u zavisnosti od toga da li nam je potrebno da znamo koje je to klase deo. Pošto nam ne trebaju takve informacije, mi ćemo koristiti `staticmethod` .

Mogli smo takođe isto postići koristeći dekoratere :

```
@staticmethod
def koliko():
    '''Prikazuje trenutnu populaciju.'''
    print('Imamo {0:d} robota'.format(Robot.populacija))
```

Dekorateri može zamisliti kao prečicu za pozivanje eksplicitne komande, kao što smo videli u ovom primeru.
Primetite da `__init__` metoda da bi pokrenula `Robot` instancu sa koristi imenom. U ovoj metodi, možemo povećati broj `populacija` za 1, jer imamo još jednog robota koji se dodaje. Takođe, primećujemo da su vrednosti `self.ime` specifične za svaki objekat, što ukazuje na prirodu promenljive objekta.
Zapamtite da morate da se obraćate varijablama i metodama istog objekta **samo** pomoću `self`. Ovo se zove *referenaca atributa*.
U ovom programu, takođe vidimo i korišćenje **docstrings** za klase i metode. Možemo pristupiti docstrings-u klase u programu pomoću `Robot.__doc__` ,a docstrings-u metode kao `Robot.kaziZdravo.__doc__` .
Baš kao i `__init__` metod, postoji još jedan poseban `__del__` metod, koji se pozivamo kada želimo da objekat umre, odnosno on se više ne koristi i koji vraća kompjuterskom sistemu komad memorije koji mu je objekat zauzimao. U ovom metodu, mi smo jednostavno smanjili broj `Robot.populacija` za 1.
`__del__` metod se koristi kada objekat nije u upotrebi i ne zna se *da li će/kada će* taj metod da se pokrene. Ako želite da ga vidimo u akciji, moramo da koristimo `del` komandu, što je upravo ono što smo ovde uradili.
Svi članovi klase su javni. Jedan izuzetak: Ako koristite članove koji čuvaju podatke sa imenima u kojima koristite *dvostruku donju crtu kao prefiks*, na primer `__privatnavar` , Pythonće da iskoristi specijalno značenje za nazive i efikasno će da je učini privatnom promenljivom.
Zbog toga, konvencija je da svaka promenljiva koja se koristi samo u okviru klase ili objekta počinje sa donjom crtom, a svi ostali nazivi su javni i mogu biti korišćeni od strane drugih klasa/objekata. Zapamtite da je ovo samo konvencija a ne osobina Python-a (osim prefiksa sa dvostrukom donjom crtom).
Napomena za C++/Java/C# programere
Svi članovi klase (uključujući i članove za podatke) su *javni* i sve metode su *virtuelne* u Python-u.

13.6 Povezivanja(Inheritance)

Jedna od glavnih prednosti objektno orijentisanog programiranja je **ponovna upotreba** koda (dela programa) i jedan od načina da se ovo postigne je mehanizam *povezivanja*. Povezivanje se najbolje može zamisliti kao određivanje odnosa *tipova i podtipova* između klasa.
Pretpostavimo da želite da napišete program koji mora da vodi evidenciju nastavnika i učenika na fakultetu. Oni imaju neke zajedničke osobine kao što su ime, starost i adresa. Oni takođe imaju specifične karakteristike kao što su plata, predavanja i odmori za nastavnike i, oznake i cene za učenike.
Možete da kreirate dve nezavisne klase za svaki tip i obradite ih, ali dodavanje nove zajedničke karakteristike bi značilo da je moramo dodati u obe nezavisne klase. Ovo bi ubrzo postalo veoma glomazno.
Bolji način bi bio da se stvori zajednička klasa `ClanoviSkole`, i onda da stvorimo nastavničke i učeničke klase *povezane* iz klase `ClanoviSkole`, tj oni će postati sub-tipovi ovog tipa (klase), pa onda možemo dodavati specifične karakteristike ovim pod-tipovima.

Postoje mnoge prednosti ovakvog pristupa. Ako mi dodajemo/menjamo nekakve funkcionalnost u klasi `ClanoviSkole`, to će se, automatski odraziti i u njenim podtipovima. Na primer, možete da dodate novo polje ID kartice za nastavnike i učenike, jednostavnim dodavanjem u klasu `ClanoviSkole`. Međutim, promene unutar podtipova ne utiču na druge podtipove. Još jedna prednost je da možete da se odnosite na objekat nastavnik ili objekat student kao objekat iz klase `ClanoviSkole`, što bi moglo da bude korisno u nekim situacijama, kao što su prebrojavanje članova škole. Ovo se zove **polimorfizam**, gde se može očekivati da će podtip biti zamenjen u svakoj situaciji u kojoj glavni tip to očekuje, odnosno objekat može da se tretira kao instanca matične klase.
Takođe primećujemo da možemo da *ponovo koristimo* kodove od matične klase i ne moramo da ih ponavljamo u različitim klasama kao što bi morali u slučaju smo koristili međusobno nezavisne klase.
Klasa `ClanoviSkole` je u ovoj situaciji poznatija po imenu *osnovna klasa* ili *superklasa*. A klase Učitelj i Student se nazivaju *izvedenim klasama* ili *podklase*.
Sada ćemo videti ovaj primer kako "radi" - pretvorimo ga u program (sačuvajte kao `inherit.py`):

```
class ClanoviSkole:
    '''Predstavlja bilo kog clana skole'''
    def __init__(self, ime, godine):
        self.ime = ime
        self.godine = godine
        print('(Inicijalizujem ClanoviSkole: {0})'.format(self.ime))

    def kazi(self):
        '''Kaze moje detalje'''
        print('Ime:"{0}" Godina:"{1}"'.format(self.ime, self.godine),
end = " ")

class Ucitelj(ClanoviSkole):
    '''Predstavlja ucitelja'''
    def __init__(self, ime, godine, zarada):
        ClanoviSkole.__init__(self, ime, godine)
        self.zarada = zarada
        print('(Inicijalizujem Ucitelj: {0})'.format(self.ime))

    def kazi(self):
        ClanoviSkole.kazi(self)
        print('Zarada: "{0:d}"'.format(self.zarada))

class Student(ClanoviSkole):
    '''Predstavlja ucenike'''
    def __init__(self, ime, godine, oznaka):
        ClanoviSkole.__init__(self, ime, godine)
        self.oznaka = oznaka
        print('(Inicijalizujem Student: {0})'.format(self.ime))

    def kazi(self):
```

```
        ClanoviSkole.kazi(self)
        print('Oznaka: "{0:d}"'.format(self.oznaka))

u = Ucitelj('Gdja. Blebetala', 40, 30000)
s = Student('Swaroop', 25, 75)

print()  # prikazuje praznu liniju

clanovi = [u, s]
for clan in clanovi:
    clan.kazi()  # radi kako za Ucitelje, tako i za Studente
```

Izlaz:

```
$ python3 inherit.py
(Inicijalizujem ClanoviSkole: Gdja. Blebetala)
(Inicijalizujem Ucitelj: Gdja. Blebetala)
(Inicijalizujem ClanoviSkole: Swaroop)
(Inicijalizujem Student: Swaroop)

Ime:"Gdja. Blebetala" Godina:"40" Zarada: "30000"
Ime:"Swaroop" Godina:"25" Oznaka: "75"
```

Kako to funkcioniše:

Da biste koristili povezivanje, moramo navesti ime bazne klase u tupli iza imena klase prilikom definisanja klase. Dalje, vidimo da `__init__` metod bazne klase se eksplicitno navodi koristeći `self` promenljivu, tako da možemo da inicijalizujemo deo bazne klase objekta. Ovo je veoma važno da zapamtite - Python neće automatski da pozove konstruktor bazne klase, morate da ga pozvati Vi na željenom mestu.

Mi takođe možemo primetiti da pozivanje metoda bazne klase se vrši tako što je klasno ime prefiks, praćen pozivom metode, a zatim dodeljivanjem `self` promenljive zajedno sa argumentima ako ih ima.

Primetimo da možemo smatrati instance od klasa `Ucitelj` ili `Student` kao instance klase `ClanoviSkole` kada koristimo `kazi` metod `ClanoviSkole` klase.

Takođe, obratite pažnju da se poziva `kazi` metod podklase, a ne metod `kazi` klase `ClanoviSkole`. Jedan od načina da se ovo shvati je taj da, Python *uvek* kreće u potragu za traženom metodom u trenutnom tipu, što, u ovom slučaju, i radi. Ako nije mogao da pronađe metod, on počinje da pretražuje metode koje pripadaju baznoj klasi, jedan po jedan, redosledom kojim su navedene u tupli u definiciji klase.

Napomena o terminologiji - ako je više od jedne klase navedeno u tupli povezivanja, onda se to naziva multi povezivanje (*multiple inheritance*).

Parametar `end` koji se koristi u `kazi()` služi za promenu - umesto da na kraju linije bude započet novi red, `print()` funkcija će da prikaže razmak.

13.7 Rezime

U ovom poglavlju smo istraživali različite aspekte klasa i objekata, kao i različite terminologije u vezi sa njima. Takođe smo videli prednosti i mane objektno orijentisanog programiranja. Python je strogo objektno-orijentisan i razumevanje ovih koncepata će vam mnogo pomoći na duže staze.

U sledećem poglavlju, mi ćemo naučiti kako da se izborimo sa ulazom/izlazom i kako da pristupite datotekama u Python-u.

14 Ulaz/Izlaz (Input/Output)

Postojaće situacije kada vaš program mora da ima nekakvu interakciju sa korisnikom. Na primer, Vi bi želeli da imate neki unos od strane korisnika, a zatim da prikažete neke rezultate te korisnikove akcije. To možemo postići pomoću `input()` i `print()` funkcija.
Za izlaz, mi možemo da koristimo različite metode `str` (string) klase. Na primer, možete da koristite `rjust` metod da prikažete string koji je poravnan sa desne strane na navedenu širinu. Pogledajte `help(str)` za više detalja.
Još jedan čest tip ulaza/izlaza je rad sa fajlovima. Sposobnost da kreira, čita i piše fajlove je od suštinskog značaja za mnoge programe i mi ćemo istražiti i ovaj aspekt u ovom poglavlju.

14.1 Ulaz od korisnika

Sačuvaj ovaj program kao `korisnik_input.py` :

```
def obrnut(tekst):
    return tekst[::-1]

def da_li_je_palindrom(tekst):
    return tekst == obrnut(tekst)

nesto = input('Ukucaj tekst: ')
if (da_li_je_palindrom(nesto)):
    print('Da, to je palindrom')
else:
    print('Ne, to nije palindrom')
```

Izlaz:

```
$ python3 korisnik_input.py
Ukucaj tekst: sir
Ne, to nije palindrom
$ python3 korisnik_input.py
Ukucaj tekst: madam
Da, to je palindrom
$ python3 korisnik_input.py
Ukucaj tekst: abrakadabra
Ne, to nije palindrom
```

Kako to funkcioniše:
Mi koristimo funkciju odsecanja da nam da tekst u obrnutom rasporedu. Već smo videli kako možemo napraviti isečke iz sekvenci korišćenjem `sekvenca[a:b]` počevši od pozicije `a` pa do pozicije `b` . Mi takođe možemo da priložimo i treći argument koji određuje *korak* (razmak) kojim se vrši isecanje. Podrazumevan korak je 1 , zbog čega nam se uvek i vraća kontinuirani deo teksta (onako kako je i unesen). Davanje negativnog koraka, odnosno, -1 će nam vratiti tekst u "rikverc" (obrnut).
`input()` funkcija može da ima string kao argument i prikazuje ga korisniku. Posle toga čeka korisnika da unesete nešto i pritisnite taster Enter. Kada korisnik unese nešto i pritisne taster Enter, `input()` će vratiti u program taj tekst koji je korisnik uneo.
Mi koristimo taj tekst tako što ga preokrenemo. Ako su originalni tekst i obrnuti tekst jednaki, onda

je taj tekst palindrom (rečenica koja se isto čita i piše i od napred i od pozadi).

Domaći zadatak

Provera da li je tekst palindrom treba da ignoriše znakove interpunkcije, razmake i veličinu slova. Na primer, "A mene, ni dogodine nema." je palindrom, ali naš trenutni program nam ne bi kazao da jeste. Možete li poboljšati gornji program da prepozna ovaj palindrom?

Savet (Ne čitaj)

Koristite tuple (možete naći listu *svih* znakova interpunkcije ovde: interpunkcija) koja sadrži sve nedozvoljene karaktere, zatim koristite test članstvo da utvrdi da li znak treba da se ukloni ili ne, odnosno da li je znak među zabranjenim = ('!','?', '.', ...).

14.2 Fajlovi

Fajlove možete otvarati i koristiti za čitanje ili pisanje stvaranjem objekata `file` klase i koristeći adekvatne njene `read` , `readline` ili `write` metode da čitate iz ili upisuje u datoteku. Sposobnost da čita ili piše u fajlu zavisi od režima koji ste odredili prilikom otvaranja datoteke. Onda, na kraju, kada završite neku manipulaciju sa datotekom, možete pozvati `close` metod da kažete Python-u da smo završili sa korišćenjem datoteke.

Primer (sačuvati kao `koristiti_fajlove.py`):

```
pesma = '''\
Programiranje je zabava
Kada je posao gotov
ako zelis da ti poso bude razbribriga
    koristi Python!
'''

f = open('pesma.txt', 'w')  # Otvara fajl za 'w' - pisanje
f.write(pesma)  # upisuje tekst u fajl
f.close()  # zatvara fajl

f = open('pesma.txt')  # Ako nismo naveli, podrazumevano je 'r' -
# citanje fajla
while True:
    linija = f.readline()
    if len(linija) == 0:  # Duzina od 0 znaci da smo dostigli EOF -
# kraj fajla
        break
    print(linija, end = ' ')
f.close()  # zatvara fajl
```

Izlaz:

```
$ python3 koristiti_fajlove.py
Programiranje je zabava
 Kada je posao gotov
 ako zelis da ti poso bude razbribriga
     koristi Python!
```

Kako to funkcioniše:

Prvo, otvaramo datoteku pomoću ugrađene `open` funkcije navodeći ime fajla i način na koji želite

da otvorite taj fajl. Režim može biti: čitanje (`'r'`), pisanje (`'w'`) ili dodavanje (`'a'`). Takođe možete odrediti da li će se čitanje, pisanje, ili dodavanje odvijati u tekstualnom modu (`'t'`) ili u binarnom modu (`'b'`). Imate zapravo mnogo više načina na raspolaganju i `help(open)` će vam dati više detalja o njima. Po defaultu, `open()` fajl će da bude `'t'`ekst fajl i otvara se u `'r'` - režimu za čitanje.
U našem primeru, prvo otvorite datoteku u režimu upisivanja teksta i koristimo `write` metod za pisanje u datoteku, a zatim smo je konačno zatvorili sa `close`.
Sledeće što smo uradili je da smo otvorili isti fajl ponovo za čitanje. Mi ne treba da navedete režim jer je "pročitaj tekst fajl" podrazumevani režim otvaranja. Mi zatim čitamo svaku liniju datoteke pomoću `readline` metode unutar petlje. Ova metoda daje kompletnu liniju uključujući znak novog reda na kraju linije. Kada se vrati *prazan* string, to znači da smo stigli na kraj fajla i mi "bežimo" iz petlje.
Podrazumevano, `print()` prikazuje tekst, ali i automatski prelazi u novi red na ekranu. Mi smo sprečili prelazak u novi red navođenjem `end = ' '` , jer će linija, koja se čita iz datoteke, već imati znak za novi red na kraju. Onda smo konačno zatvorili datoteku sa `close`.
Sada, proverite sadržaj `pesma.txt` datoteke da biste potvrdili da je program zaista pisao i čitaju iz tog fajla.

14.3 Pickle

Python poseduje standardni modul koji se zove `pickle`, pomoću koga možete da uskladištite **bilo kakav** Python-ov objekat u fajlu i odatle ga koristiti kasnije. Ovo se zove *perzistentno* čuvanje objekata.
Primer (sačuvati kao `picklovanje.py`):

```
import pickle

# ime fajla u kom zelimo da cuvamo objekat
fajllistekupovine = 'listakupovine.data'
# lista stvari za kupovinu
listakupovine = ['jabuka', 'mango', 'sargarepa']

# Sacuvaj u fajl
f = open(fajllistekupovine, 'wb')
pickle.dump(listakupovine, f)  # stavlja objekat u fajl
f.close()

del listakupovine  # unistava listakupovine promenljivu

# Ucitava nazad iz fajla
f = open(fajllistekupovine, 'rb')
sacuvanalista = pickle.load(f)  # ucitava objekat iz fajla
print(sacuvanalista)
```

Izlaz:

```
$ python3 picklovanje.py
['jabuka', 'mango', 'sargarepa']
```

Kako to funkcioniše:

Da bi memorisali neki objekat u fajlu, prvo moramo da "otvorimo" taj fajl sa `open` u `'b'` - binarnom režimu i `'w'` -upisivanje režimu, a zatim moramo pozvati `dump` funkciju `pickle` modula. Ovaj proces se naziva *pickling*.
Dalje, mi smo preuzeli objekat pomoću `load` funkcije `pickle` modula, koja vraća objekat. Ovaj proces se naziva *unpickling*.

14.4 Rezime

Razmatrali smo razne vrste ulaza/izlaza, a takođe i rad sa fajlovima i korišćenja modula `pickle`.
U sledećem poglavlju mi ćemo obrađivati koncept hvatanja grešaka (exceptions).

15 Exception ("Hvatanje grešaka")

Izuzeci se javljaju kada se pojave određene *izuzetne* situacije u vašem programu. Na primer, šta ako želite da pročitate datoteku a datoteka ne postoji? Ili šta ako je slučajno izbrišemo u trenutku kada je program pokrenut? Sa takvim situacijama se borimo koristeći **exceptions**.
Slično tome, šta ako u vašem programu postoje neke neispravne komande? Njega će obrađivati Python, i kada naleti na neispravan deo koda, on će "dići ruke" (**raises**) i jednostavno vam reći da postoji **greška**.

15.1 Greške

Zamislite jednostavnu `print` funkciju. Šta ako smo pogrešno napisali `print` kao `Print` ? Obratite pažnju na velika i mala slova u rečima. U ovom slučaju, Python *raises* (podiže) sintaksnu grešku.

```
>>> Print('Pozdrav Svima!')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>> print('Pozdrav Svima!')
Pozdrav Svima!
```

Primetite da je `NameError` greška podignuta (raised) a prikazano je i mesto gde je otkrivena greška. To je ono što "hvatač" grešaka (*error handler*) za ovu grešku uradi.

15.2 Izuzeci (exceptions)

Mi ćemo sada **pokušati** da čitamo ulaz od korisnika. Kada Vas program bude pitao za unos pritisnite `ctrl-d` (ako koristite Windows - `ctrl - z`), pa vidite šta se dešava.

```
>>> s = input('Ukucaj nesto--> ')
Ukucaj nesto-->
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Python podiže grešku koja se zove `EOFError` i koja u suštini znači da je našao *simbol kraja datoteke* (koji predstavlja `ctrl-d`) u trenutku kada se to ne očekuje.

15.3 Rukovanje ca exceptions-ima

Mi možemo da kontrolišemo exceptions pomoću komandi `try..except`. U suštini mi stavljamo uobičajene komande unutar try bloka i stavio sve naše komande za kontrolu greške u bloku except.
Primer (sačuvati kao `try_except.py`):

```
try:
    tekst = input('Ukucaj nesto --> ')
except EOFError:
    print('Zasto si mi uradio EOF?')
except KeyboardInterrupt:
    print('Ti si ponistio operaciju.')
else:
```

```
    print('Ti si uneo {0}'.format(tekst))
```

Izlaz:

```
$ python3 try_except.py
Ukucaj nesto -->       # pritisnite ctrl -d
Zasto si mi uradio EOF?
$ python3 try_except.py
Ukucaj nesto -->    # pritisnite ctrl -c
Ti si ponistio operaciju.
$ python3 try_except.py
Ukucaj nesto --> nema exceptiona
Ti si uneo nema exceptiona
```

Kako to funkcioniše:

Mi smo stavili sve izjave koje mogu da dovedu do izuzetaka/grešaka unutar `try` bloka, a zatim stavite rešenja za odgovarajuće greške/izuzetake u `except` klauzulu/blok. `except` klauzula može da obradi jednu određenu grešku ili izuzetak, ili malo veću listu grešaka/izuzetaka. Ako nismo naveli imena grešaka ili izuzetka ovaj blok će kontrolisati *sve* greške i izuzetke koji se pojave.

Imajte na umu da mora da postoji barem jedna `except` klauzula u vezi sa svakim `try` blokom. Inače, u čemu bi bila poenta `try` bloka?

Ako postoje greške ili izuzetci koje nismo naveli, onda će standardni Python handler biti pozvan, i on će da zaustavi izvršavanje programa i prikazati poruku o greški. Već smo to ranije videli u akciji.

Takođe možete da imate `else` blok povezan sa `try..except` blokom. `else` blok će se izvršiti ako se nije desio izuzetak/greška.

U sledećem primeru ćemo videti kako da se "uhvati" objekat izuzetaka, tako da možemo da dobijemo neke dodatne informacije.

15.4 Podizanje Exception-a

Možete da *podignete* izuzetke pomoću `raise` komande dajući ime greške/izuzetka i `exception` objekta koji treba da bude *izbačen*.

Greška ili izuzetak koji želite podići treba da bude klasa koja, direktno ili indirektno mora biti izvedena klasa iz `Exception` klase.

Primer (sačuvati kao `raising.py`):

```
class IzuzetakZaKratkiUnos(Exception):
    '''Klasa definisana od strane korisnika.'''
    def __init__(self, duzina, najmanje):
        Exception.__init__(self)
        self.duzina = duzina
        self.najmanje = najmanje

try:
    tekst = input('Ukucaj nesto --> ')
    if len(tekst) < 3:
        raise IzuzetakZaKratkiUnos(len(tekst), 3)
    # Neki drugi rad moze biti definisan na ovom mestu,
    # Na uobicajen nacin
```

```
except EOFError:
    print('Zasto si mi uradio EOF?')
except IzuzetakZaKratkiUnos as ex:
    print('IzuzetakZaKratkiUnos: Unos je {0} karaktera dug, ocekivano
je najmanje {1}'\
          .format(ex.duzina, ex.najmanje))
else:
    print('Nijedan exception nije podignut.')
```

Izlaz:

```
$ python3 raising.py
Ukucaj nesto --> a
IzuzetakZaKratkiUnos: Unos je 1 karaktera dug, ocekivano je najmanje 3
$ python3 raising.py
Ukucaj nesto --> asd
Nijedan exception nije podignut.
```

Kako to funkcioniše:
Ovde mi stvaramo sopstveni tip izuzetaka. Ovaj novi tip izuzetaka se zove `IzuzetakZaKratkiUnos`. On ima dva polja - `duzina` je dužina datog ulaza i `najmanje` koja predstavlja minimalnu dužinu koju program očekuje.
U `except` klauzuli, mi smo naznačili klasu greške koja će biti memorisana kao promenljiva sa imenom ex, koja će da sadrži odgovarajući objekat greške/izuzetka. Ovo je analogija parametrima i argumentima prilikom pozivanja funkcija. U okviru ove konkretne `except` klauzule, mi koristimo polja `duzina` i `najmanje` od objekta exception da bi prikazali odgovarajuću poruku korisniku.

15.5 Try .. Finally

Pretpostavimo da učitate datoteku u Vaš program. Kako bi se osigurali da je objekat fajla pravilno zatvoren ako nije nijedan izuzetak podignut? Ovo se može uraditi pomoću `finally` bloka. Imajte na umu da možete da koristite `except` klauzulu zajedno sa `finally` blokom za odgovarajući `try` blok. Morate da ugradite jedan u drugog, ako želite da koristite oba.
Sačuvajte kao `finally.py` :

```
import time

try:
    f = open('pesma.txt')
    while True:  # Nas uobicajeni nacin citanja fajlova
        linija = f.readline()
        if len(linija) == 0:
            break
        print(linija, end = ' ')
        time.sleep(2)  # Da budemo sigurni da ce da radi neko vreme
except KeyboardInterrupt:
    print('!! Vi ste prekinuli citanje fajla.')
finally:
    f.close()
```

```
    print('(Cistimo: Zatvaramo fajl)')
```

Izlaz:

```
$ python3 finally.py
Programiranje je zabava
 Kada je posao gotov
 ako zelis da ti poso bude razbribriga
 !! Vi ste prekinuli citanje fajla.
 (Cistimo: Zatvaramo fajl)
```

Kako to funkcioniše:
Mi radimo uobičajeno čitanje datoteka, ali smo proizvoljno uveli pauzu od 2 sekunde nakon prikaza svake linije, koristeći `time.sleep` funkciju, tako da se program pokreće sporije (Python je, inače, vrlo brz). Kada program pokrenete i dok se on izvršava, pritisnite `ctrl-c` da prekinete/otkažete izvršavanje programa.
Vidimo da je `KeyboardInterrupt` izuzetak izbačen i program se završava. Međutim, pre nego izađemo iz programa, finally-blok kreće da se izvršava i objekat datoteke se uvek zatvara.

15.6 with komanda

Dobijanje informacija u `try` bloku, a kasnije oslobađanje tih informacija u `finally` bloku je najčešće korišćen scenario rada. Ali, tu je i `with` naredba koja omogućava da se to uradi na jednostavniji način:
Sačuvajte kao `koristiti_with.py` :

```
with open("pesma.txt") as f:
    for linija in f:
        print(linija, end = ' ')
```

Kako to funkcioniše:
Izlaz bi trebalo da bude isti kao i u prethodnom primeru. Razlika je u tome što ovde koristimo `open` funkciju sa `with` naredbom - mi ostavimo da nam zatvaranje datoteke automatski uradi `with open` .
Šta se dešava iza scene - postoji protokol koji koristi `with` naredbu. On preuzima objekat koji vraća `open` komanda, nazovimo ga "`file`" u ovom slučaju.
Taj protokol *uvek* poziva `file.__enter__` funkciju pre početka bloka naredbi ispod njega, i *uvek* poziva `file.__exit__` nakon završetka bloka naredbi.
Dakle, kod koji bi smo morali napisati u `finally` bloku, će biti odrađen automatski od strane `__exit__` metode. To je ono što nam pomaže da se izbegne korišćenje eksplicitnih `try..finally` komandi konstantno unutar programa.
Veće razlaganje na ovu temu je izvan okvira ove knjige, pa pogledajte PEP 343 za sveobuhvatno objašnjenje.

15.7 Rezime

Objasnili smo korišćenje `try..except` i `try..finally` komandi. Takođe, videli smo kako da kreirate sopstvene vrste exception i kako da se raise exception.
U sledećem poglavlju, mi ćemo istraživati Python-ove standardne biblioteke.

16 Standardne biblioteke

Python-ove standardne biblioteke sadrže veliki broj korisnih modula i deo su svake standardne Python instalacije. Važno je da se upoznate sa Python-ovim standardnim bibliotekama, jer se mnogi problemi mogu brže rešiti, ako ste upoznati sa spektrom stvari koje ove biblioteke mogu da pruže.
Mi ćemo istražiti neke od najčešće korišćenih modula u ovim bibliotekama. Možete naći kompletne podatke za sve modula u standardnim bibliotekama u odeljku "Library Reference" unutar dokumentacije koja dolazi sa instalacijom Python-a.
Hajde da istražimo nekoliko korisnih modula.

Pažnja

Ako vam se teme u ovom poglavlju čine suviše napredne, možete da preskočite ovo poglavlje. Međutim, ja strogo preporučujem da se vratite ovom poglavlju, kada Vam postane blisko programiranje uz pomoć Python-a.

16.1 sys modul

`sys` modul sadrži specifične sistemske funkcionalnosti. Već smo videli da lista `sys.argv` sadrži argumente komandne linije.
Pretpostavimo da želite da proverite koja verzija Python-a se koristi, tako da, recimo, mi želimo da budemo sigurni da koristimo najmanje 3 verziju programa. `sys` modul nam daje takvu mogućnost.

```
$ python3
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=3, micro=0, releaselevel='final',
serial=0)
>>> sys.version_info.major >= 3
True
```

Kako to funkcioniše:
`sys` modul ima tuple `version_info`, koja nam daje informacije o verziji programa. Prvi podatak je glavna verzija. Mi možemo to da proverimo iz razloga, kao na primer, da obezbedimo da program radi samo pod Python 3.0:
Sačuvajte kao `proveraverzije.py` :

```
import sys, warnings
if sys.version_info.major < 3:
    warnings.warn("Potreban Vam je Python 3.0 da bi pokrenuli ovaj
program",
                  RuntimeWarning)
else:
    print('Nastaviti normalno')
```

Izlaz:

```
$ python2.7 proveraverzije.py
proveraverzije.py:6: RuntimeWarning: Potreban Vam je Python 3.0 da bi
pokrenuli ovaj program
           RuntimeWarning)
$ python3 proveraverzije.py
```

```
Nastaviti normalno
```

Kako to funkcioniše:
Mi koristimo drugi modul iz standardne biblioteke pod nazivom `warnings` koja se koristi za prikazivanje upozorenja krajnjem korisniku. Ako broj verzije Python-a nije barem 3, mi smo prikazali odgovarajuće upozorenje.

16.2 logging modul

Šta ako ste želeli da imate neke poruke koje bi pomogle u otklanjanju grešaka ili neke važne poruke koje bi se negde čuvale, tako da možete da proverite kako i šta vaš program pokreće/radi i da li radi kako ste očekivali? Kako bi "zadržali negde" ove poruke? Ovo se može postići korišćenjem `logging` modula.
Sačuvajte kao `koristiti_logovanje.py` :

```
import os, platform, logging

if platform.platform().startswith('Windows'):
    log_fajl = os.path.join(os.getenv('HOMEDRIVE'),
os.getenv('HOMEPATH'), 'test.log')
else:
    log_fajl = os.path.join(os.getenv('HOME'), 'test.log')

print("Logujem u", log_fajl)

logging.basicConfig(
    level = logging.DEBUG,
    format = '%(asctime)s : %(levelname)s : %(message)s',
    filename = log_fajl,
    filemode = 'w',
)

logging.debug("Pocetak programa")
logging.info("Radim nesto")
logging.warning("A sad umirem")
```

Izlaz:

```
$ python3 koristiti_logovanje.py
Logujem u C:\Documents and Settings\xlure\test.log
```

Ako bismo proverili sadržaj `test.log` fajla, on će izgledati ovako:

```
2013-04-10 22:20:43,406 : DEBUG : Pocetak programa
2013-04-10 22:20:43,421 : INFO : Radim nesto
2013-04-10 22:20:43,421 : WARNING : A sad umirem
```

Kako to funkcioniše:
Mi koristimo tri modula iz standardne biblioteke - `os` za interakciju sa operativnim sistemom, `platform` modul za informacije o platformi, odnosno operativnom sistemu i `logging` modul za *logovanje* informacija.

Prvo, mi proveravamo koji operativni sistem se koristi proverom vraćenog stringa iz modula `platform.platform()` (za više informacija pogledajte `import platform; help(platform)`). Ako je u pitanju Windows, mi saznajemo primarni disk, matični folder i naziv fajla gde želimo da skladištimo informacije. Spajanjem ova tri dela, mi smo dobili punu lokaciju datoteke. Za druge platforme, treba samo da znamo Home folder korisnika i mi dobijamo punu lokaciju datoteke.
Mi koristimo `os.path.join()` da sastavim ove tri dela i dobijemo putanju do lokacije. Razlog da se koristi specijalna funkcija, a ne sabiranje stringova je zato što će ova funkcija da obezbediti lokaciju u odgovarajućem formatu, koji se očekuje od strane operativnog sistema.
Mi smo konfigurisali `logging` modul da piše poruke u određenom formatu u datoteku koju smo naveli.
Konačno, mi definišemo poruke koje su namenjene ili otklanjanju grešaka, informacije, upozorenja ili čak kritične poruke. Jednom kada pokrenete program, možete proveriti ovaj fajl i mi ćemo znati šta se desilo tokom rada programa, iako informacije nisu prikazane korisniku programa.

16.3 Moduli nedelje serije

Ima mnogo više toga da se istraži u standardnim bibliotekama, kao što je debagovanje, rad sa opcijama komandne linije , regular expressions i tako dalje.
Najbolji način da dodatno istražite standardnu biblioteku je da pročitate Doug Hellmann-ovu odličnu Python modul nedelje ili isčitavanjem Python dokumentacije.

16.4 Rezime

U ovom poglavlju obrađivali smo neke od funkcionalnosti nekih modula unutar Python standardne biblioteke. Preporučuje se da pregledate dokumentaciju Python standardne biblioteke da bi dobili predstavu o svim modulima koji su dostupni.
U sledećem poglavlju ćemo pokriti različite aspekte Python-a, što će malo *kompletirati* naš put kroz Python programiranje.

17 Malo više...

Do sada smo obradili većinu različitih aspekata Python-a koje ćete koristiti. U ovom poglavlju ćemo pokriti još neke aspekte, kako bi zaokružili naše znanje o Python-u.

17.1 Igranje sa tupleima

Da li ste ikada poželeli da možete da dobijete dve različite vrednosti iz funkcije? Možete. Sve što treba da uradite je da koristite tuple.

```
>>> def daje_detalje_greske():
    return(2, 'drugi detalj greske')

>>> brojgreske, stringgreske = daje_detalje_greske()
>>> brojgreske
2
>>> stringgreske
'drugi detalj greske'
```

Primetimo da korišćenje `a, b = <neka ekspresija>` tumači rezultat izraza kao tuple sa dve vrednosti.
Ako želite da rezultati budu tumačeni kao `(a, <sve ostalo>)` , onda samo treba da koristimo zvezdicu, baš kao što smo to radili u parametrima funkcije:

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

To takođe znači da je najbrži način da se zameni vrednost dve promenljive u Python-u je:

```
>>> a = 5; b = 8
>>> a, b = b, a
>>> a, b
(8, 5)
```

17.2 Posebne metode

Postoje određene metode, kao što su `__init__` i `__del__` koje imaju poseban značaj u klasama.
Posebni metodi se koriste da imitiraju ponašanje određenih ugrađenih tipova. Na primer, ako želite da koristite `x[kljuc]` operaciju indeksiranja za Vašu klasu (baš kao da je koristite za liste i tuple), onda, sve što treba da uradite je, da implementirate `__getitem__()` i vaš posao je završen. Ako mislite o tome, to je upravo ono što Python radi za `list` klasu!
Neke korisne posebne metode su navedene u sledećoj tabeli. Ako želite da znate o svim specijalnim metodama, pogledajte uputstvo.
`__init__(self, ...)` - Ova metoda se poziva baš kada se novokreirani objekat vraća za upotrebu.
`__del__(self)` - Pozvia se neposredno pre nego što je objekat uništen.
`__str__(self)` - Poziva se kada koristimo `print` funkciju ili `str()` funkciju.
`__lt__(self, drugi)` - Poziva se kada se koristi "*manje od*" operator (<). Slično ovom, postoje

posebni metodi za sve operatere (+, >, itd).
`__getitem__(self, kljuc)` - Poziva se kada se koristi `x[kljuc]` operacija indeksiranja.
`__len__(self)` - Poziva se kada se koristi ugrađena `len()` funkcija za sekvence objekata.

17.3 Blokovi sačinjeni od samo jedne naredbe

Videli smo da se svaki blok naredbi odvaja od ostatka sa sopstvenim nivoom uvlačenja. Pa, tu postoji jedna začkoljica. Ako vaš blok naredbi sadrži samo jednu komandu, onda taj blok možete da definišete na istoj liniji, recimo, uslovne komande ili neredbe petlje. Sledeći primer bi trebalo to da malo pojasni:

```
>>> zastava = True
>>> if zastava: print('Da')

Da
```

Primetite da je jedna komanda korišćena u-mestu, a ne kao poseban blok. Iako možete koristiti ovo da bi prilikom pisanja vaš program bio *kraći*, preporučujem izbegavanje ove prečice, osim za proveru grešaka, uglavnom zbog toga što će biti mnogo lakše da dodate ekstra komandu ako koristite odgovarajuće uvlačenje blokova.

17.4 Lambda forme

`lambda` komanda se koristi za kreiranje novih objekata funkcije. U suštini, `lambda` uzima parametar praćen samo jednim izrazom koji postaje telo funkcije i vrednosti ovog izraza je vrednost koju nam vraća ova nova funkcija.
Primer (sačuvajte kao `lambda.py`):

```
tacke = [{'x' : 2, 'y' : 3}, {'x' : 4, 'y' : 1}]
tacke.sort(key = lambda i : i['y'])
print(tacke)
```

Izlaz:

```
[{'y': 1, 'x': 4}, {'y': 3, 'x': 2}]
```

Kako to funkcioniše:
Primetimo da `sort` metoda za `list`-u, može imati `key` parametar koji određuje kako se lista sortira (obično mi samo koristimo rastući ili opadajući redosled). U našem slučaju, mi želimo da uradimo prilagođeno sortiranje, i za to bi nam bilo potrebno da napišemo funkciju, ali umesto pisanja posebnog `def` bloka funkcije koja bi se koristila samo na tom jednom mestu, mi koristimo `lambda` ekspresiju da bi kreirali tu novu funkciju.

17.5 Kompresovanje lista

Kompresovanje lista se koristi da bi izgradili novu listu iz neke već postojeće liste. Pretpostavimo da imate listu brojeva i želite da dobijete odgovarajuću listu sa svim brojevima pomnoženim sa 2, samo ako je broj veći od 2. Kompresovanje lista je idealan za ovakvu situaciju.
Primer (sačuvati kao `kompresovanje_lista.py`):

```
prvalista = [2, 3, 4]
drugalista = [2 * i for i in prvalista if i > 2]
```

```
print(drugalista)
```

Izlaz:

```
$ python3 kompresovanje_lista.py
[6, 8]
```

Kako to funkcioniše:
Ovde smo izgradili novu listu navodeći koju manipulaciju želimo da uradimo (`2*i`), ukoliko je neki uslov zadovoljen (`if i > 2`). Imajte na umu da originalna lista ostaje nepromenjena.
Prednost korišćenja kompresovanja lista je da smanjuje količinu opšte namenskog koda potrebnog prilikom pisanja petlje koja bi nam koristila da obradimo svaki elemenat iz liste i da bi ga sačuvala u novoj listi.

17.6 Slanje tuple i rečnika funkcijama

Postoji poseban način da funkcija prima parametare u vidu tuplea ili rečnika koristeći oznake * i ** kao prefiks. Ovo je korisno kada treba poslati promenljivi broj argumenata za funkciju.

```
>>> def sumastepena(stepen, *argumenti):
    '''Vraca sumu svih argumenata stepenovanih

na specijalni broj'''
    totalno = 0
    for i in argumenti:
        totalno += pow(i, stepen)
    return totalno

>>> sumastepena(2, 3, 4)
25
>>> sumastepena(2, 10)
100
```

Zato što imamo prefiks * ispred `argumenti` promenljive, svi dodatni argumenti koje dajemo funkciji se čuvaju u `argumenti` kao tuple. Ako se, umeso *, koristi ** prefiks, dodatni parametri će se smatrati kao da su ključ/vrednost parovi unutar rečnika.

17.7 assert naredba

`assert` naredba se koristi da bi se potvrdilo da li je nešto istina. Na primer, ako ste sigurni da ćete imati barem jedan element u listi koju koristite, a želite da proverite, i da podignete grešku ako to nije istina, onda je `assert` komanda idealna u toj situaciji. Ako assert komanda ne prođe, `AssertionError` je podignut.

```
>>> mojalista = ['stvar']
>>> assert len(mojalista) >= 1
>>> mojalista.pop()
'stvar'
>>> mojalista
[]
```

```
>>> assert len(mojalista) >= 1
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    assert len(mojalista) >= 1
AssertionError
```

`assert` komandu treba promišljeno koristi. U većini slučajeva, bolje je uhvati izuzetke, ili rukovati sa greškama ili prikazati poruku o grešci korisniku, a zatim izaći iz programa.

17.8 Escape Sequences

Pretpostavimo, želite da imate string koji treba da sadrži znak za jedan citat ('). Kako ćete načiniti ovaj string? Na primer, string `What's your name?` . Ne možete napisati `'What's your name?'` , jer će Python biti zbunjeni - gde string počinje, a gde se završava. Dakle, vi ćete morati da, na neki način, ukažete da ovaj znak za citat ne ukazuje na kraj stringa. Ovo se može uraditi uz pomoć onoga što se zove *escape sequence*. Vi treba da odredite taj jedan citat kao `\'` - primetite obrnutu kosu crtu. Sada možete da napišete ovaj string kao `'What\'s your name?'`.
Drugi način navođenja ovakvih specifičnih stringova bi bio `"What's your name?"` , odnosno korišćenjem navodnika. Slično tome, morate da koristite escape sequence za korišćenje navodnika unutar navodnika. Takođe, morate da ukažete na samu obrnutu kosu crtu koristeći escape sequence `\\` .
Šta ako ste želeli da odredite string u dve linije? Jedan način je da se koristite trostruke citate, kao što je prikazano ranije, ili možete da koristite escape sequence za novi red karaktera - `\n` koji će da ukaže na početak nove linije. Primer je `Ovo je prva linija\nOvo je druga linija` . Još jedna korisna escape sequence je poznatija kao tab - `\t` . Postoji mnogo više escape sequence, ali sam ovde pomenuo samo one najkorisnije u radu.
Jedna stvar koju treba napomenuti na ovom mestu je da je u stringu, jedna obrnuta kosa crta na kraju linije označava da se string nastavlja u sledećem redu, ali ne i da je dodat novi red. Na primer:

```
"Ovo je prva recenica. \
Ovo je druga recenica."
```

je ekvivalentno sa:

```
"Ovo je prva recenica. Ovo je druga recenica"
```

17.8.1 Raw String

Ako vam je potrebno da navedete neki string u kom se ne zahteva nikakva posebna obrada, kao što su escape sequence, onda ono što Vam je potrebno je da taj string navedete kao *Raw String* tj. kao prefiks `r` ili `R` pre stringa. Primer je `r"Znak za novi red se oznacava  \n"` .
Napomena za korisnike Regular Expression-a
Uvek koristite Raw String kada se bavite sa Regular Expression. Inače će vam trebati mnogo obrnutih kosih crta. Na primer, povratna referenca može da se poziva `'\\1'` ili `r'\1'` .

17.9 Rezime

Ovde smo pokrili još neke karakteristike Python-a, a ipak nismo pokrili baš sve funkcionalnosti koje postoje. Međutim, u ovom trenutku, mi smo pokrili većinu onoga što ćete ikada koristiti u praksi. To je dovoljno da počnete sa izradom onih programa kojih želite.
U sledećem poglavlju ćemo razgovarati o tome kako da ubuduće istražujete Python.

18 Kojim putem dalje?

Ako ste pažljivo do sada čitali ovu knjigu, i praktikovali pisanje puno programa, onda mora da Vam je već postalo prijatno i ugodno da radite sa Python-om. Verovatno ste napravili neke programe kako bi ste isprobavali različite stvari i da izoštrite svoje veštine. Ako niste to već pokušali, znajte: to vam treba. Sada se postavlja pitanje: "Šta dalje?".
Predložio bih da se pozabavite ovim problemom:
Napravite svoj program koji će se izvršavati putem komandne linije. Neka to bude *adresar*, u kojem možete da pregledate, dodajete, menjate, brišete ili tražite svoje kontakte, kao što su prijatelji, porodica i kolege i njihove informacije, kao što su imejl adresa i/ili broj telefona. Detalji moraju biti čuvani negde, da bi se mogli upotrebljavati i kasnije (bilo kada).
To je prilično lako ako zamišljate to, u okviru svih onih raznih stvari koje smo objašnjavali do sada. Ako ipak želite uputstva o tome kako da nastavite, onda evo malog saveta.
Savet (Nemojte da čitte, a da niste već pokušali da rešite problem)
Kreirajte klasu koja će predstavljati informacije o osobi. Koristite rečnik za skladištenje objekata osoba sa ključem koji je njihovo ime. Koristite modul Pickle da bi trajno skladištili objekte na čvrstom disku. Koristite ugrađene metode rečnika za dodavanje, brisanje i menjanje informacija o osobama.
Jednom kada uspete da uradite ovo, možete za sebe da tvrdite da ste Python programer. A tada, u istom trenu, mi pošaljite poruku zahvalnosti, za ovu lepu knjigu ;-) (poruka ide originalnom autoru, a ne prevodiocu :p). Ovaj drugi korak je opcion, ali se preporučuje. Takođe, molimo Vas da razmislite o kupovini štampanog primerka za podršku daljem razvoju ove knjige.
Ako Vam je rešavanje zadatka i pravljenje ovog programa bilo lako, evo još jednog:
Implementirajte komandu zamene . Ova komanda će zameniti jedan string sa drugim u listi datoteka koje su joj priložene.
Naredba zamene može biti jednostavna ili sofisticirana, kako god Vi želite, od proste zamene stringa, do potrage za šablonima (regular expressions).
Nakon toga, u nastavku knjige su neki od načina da nastavite Vaš put kroz Python:

18.1 Primer koda

Najbolji način da se nauči programski jezik je da se piše mnogo koda i pročitao mnogo koda:

- Python Cookbook je izuzetno vredna zbirka recepata i saveta o tome kako da se reše određene vrste problema koristeći Python. Ova stavka je pod "mora da se pročita" za svakog Python korisnika.
- Python Module of the Week je još jedno odlično "mora da se pročita" uputstvo za Python-ove standardne biblioteke.

18.2 Pitanja i odgovori

- Zvanični Python Raditi i Ne raditi
- Zvanična Python pitanja i odgovori
- Norvig-ov spisak retko postavljanih pitanja
- Python Interview Q & A
- StackOverflow pitanja u vezi sa python-om

18.3 Tutorijali

- Awaretek's sveobuhvatni spisak Python tutorijala

18.4 Video

- PyVideo

18.5 Diskusija

Ako ste zaglavljeni sa nekim Python problemom, i ne znate koga da pitate, onda je python-tutor list najbolje mesto da postavite pitanje.
Osigurajte se tako što ćete da uradite svoj domaći zadatak i prvo sami pokušati rešavanje problema.

18.6 Vesti

Ako želite da saznate šta je najnovije u svetu Python-a, možete da pratite Official Python Planet.

18.7 Instaliranje biblioteka

Postoji veliki broj Python biblioteka otvorenog koda u Python Package Index koje možete koristiti u svojim programima.
Da biste instalirali i koristili ove biblioteke, možete da koristite pip.

18.8 Grafički softver

Pretpostavimo da želite da kreirate sopstvene grafičke programe pomoću Python-a. Ovo se može uraditi pomoću GUI (grafički korisnički interfejs) biblioteka i njihovih veza sa Python-om. Veze (Bindings) su ono što nam omogućava da pišemo programe u Python-u i koristimo biblioteke koje su pisane u C ili C++ ili nekim drugim jezicima.
Postoji mnogo izbora za izradu GUI koristeći Python:
Kivy - http://kivy.org
PyGTK - Ovo je veza Python-a sa GTK+ toolkit koji je temelj na kome se gradi GNOME. GTK+ ima mnogo začkoljica u upotrebi, ali kada postanete dobro upoznati sa njim, možete brzo kreirati GUI aplikacije. Glade grafički interfejs dizajner je nezaobilazan. Dokumentacija se još uvek poboljšava. GTK+ dobro radi na Linux-y ali port za Windows je nekompletan. Možete da napravite kako slobodan, tako i vlasnički softver koristeći GTK+. Za početak, pročitajte PyGTK tutorial.
PyQt - Ovo je veza Python-a sa Qt toolkit-om koji je temelj na kojem se izgradio KDE. Qt je izuzetno jednostavan za korišćenje i vrlo moćan, posebno zbog Qt Designer-a i neverovatne Qt dokumentacije. PyQt je besplatan, ako želite da napravite program otvorenog koda (GPL licenca), a morate da ga kupite ako želite da napravite vlasnički softver zatvorenog koda. Počevši sa Qt 4.5 možete da ga koristiti za kreiranje ne-GPL softvera. Za početak, pročitajte PyQt tutorial ili PyQt book.
wxPython - Ovo je veza Python-a sa wxWidgets toolkit-om. Da bi koristili wxPython mora da postoji jaka volja za njegovim učenjem. Međutim, on je veoma prenosiv i radi na Linux-y, Windows-y, Mac-y i još u mnogim embedded platformama. Postoji mnogo IDE-a na raspolaganju za wxPython koji sadrže i alate za GUI dizajniranje kao što su SPE (Stani's Python Editor) i wxGlade GUI alat. Možete kreirati besplatan, kao i vlasnički softver pomoću wxPython-a. Za početak, pročitajte wxPython tutorial.

18.8.1 Rezime GUI alata

Za više opcija, pogledajte GuiProgramming wiki page at the official python website.
Nažalost, ne postoji jedan standardni GUI alat za Python. Predlažem da izaberete jedan od gore navedenih alata u zavisnosti od situacije u kojoj su Vam potrebni. Prvi faktor je da li ste spremni da

platite da koristite bilo koji od GUI alata. Drugi faktor je da li želite da se program pokreće samo na Windows-u ili Mac-u i Linux-u ili je pisan za sve nabrojane. Treći faktor, ako je odabran Linux, je da li je Vaš program namenjen KDE ili GNOME korisnicima.
Za detaljniji i sveobuhvatnu analizu, vidi stranu 26 The Python Papers, Volume 3, Issue 1.

18.9 Razne implementacije

Obično postoje dva dela programskih jezika - sam jezik i softver. Jezik je *kako* pišete nešto. Softver je *ono što* zapravo pokreće naše programe.
Mi smo pokretali naše programe korišćenjem softvera *CPython*. Naziva se CPython jer je napisan u C jeziku i on je *Classical Python interpreter*.
Postoji i drugi softver koji može da pokreće Vaše Python programe:
Jython - Implementacija Python-a koja radi na Java platformi. To znači da možete da koristite Java biblioteke i klase u Python jeziku i obrnuto.
IronPython - Implementacija Python-a koja radi na .NET platformi. To znači da možete da koristite .NET biblioteke i klase u Python jeziku i obrnuto.
PyPy - Implementacija Python-a napisana u Python-u! Ovo je istraživački projekat da bi se brzo i lako poboljšavao interpreter jer je sam interpreter napisan u dinamičnom jeziku (za razliku od statičnih jezicika kao što su C, Java ili C# u tri gornje implementacije).
Stackless Python - Implementacija Python-a koja je specijalizovana za thread-based poboljšanje performansi.
Postoje i mnogi drugi poput CLPython - implementacija napisana u Common Lisp-u i IronMonkey koji je port IronPython-a prilagođen da radi preko JavaScript interpretera, što bi moglo da znači da možete da koristite Python (umesto JavaScript-a) da pišete programe za web pretraživače ("Ajax").
Svaka od ovih implementacija ima svoje specijalizovane oblasti u kojima su korisne.

18.10 Funkcionalno programiranje (za napredne čitaoce)

Kada počnete da pišete veće programe, definitivno treba da naučite više o funkcionalnom pristupu programiranju nasuprot pristupu programiranju zasnovanom na klasama koje smo naučili u poglaljvlju o objektno-orijentisanom programiranju:

- Functional Programming Howto by A.M. Kuchling
- Functional programming chapter in 'Dive Into Python' book
- Functional Programming with Python presentation

18.11 Rezime

Sada smo došli do kraja ove knjige, ali, kako kažu, *ovo je početak kraja*. Sada ste strastveni Python korisnik i vi ste bez sumnje spremani da rešavate mnoge probleme koristeći Python. Možete početi sa automatizacijom računara kako bi ste učiniti različite vrste prethodno nezamislivih stvari ili napisati sopstvene igre i mnogo, mnogo više. Dakle, počnite!

19 FLOSS

Free/Libre and Open Source Software (Slobodan/Besplatan i Softver Otvorenog Koda), ukratko, FLOSS se zasniva na konceptu zajednice, koja se i sama zasniva na konceptu deljenja, a posebno razmene znanja. FLOSS je besplatan za korišćenje, modifikaciju i redistribuciju.
Ako ste pročitali ovu knjigu, onda ste već upoznati sa FLOSS jer ste svo vreme koristili **Python**, a Python je softver otvorenog koda!
Evo nekih primera FLOSS-a, da bih Vam dao ideju o spektru stvari koje zajednica deljenjem i građenjem može da napravi:
Linux - Ovo je FLOSS kernel, koji se koristi u GNU/Linux operativnim sistemima. Linux kernel je stvorio Linus Torvalds još kad je bio student. Android je zasnovan na Linuksu. Bilo koji sajt koji posetite u današnje vreme uglavnom će biti pokrenut na Linuksu.
Ubuntu - To je distribucija vođena snažnom zajednicom, pod pokroviteljstvom tvrtke Canonical, i to je jedna od najpopularnijih Linuks distribucija danas. Ona vam omogućava da instalirate obilje FLOSS-a koji je na raspolaganju i sve to na jedan način koji je lak za korišćenje i lak za instaliranje. Najbolje od svega, možete samo da restartujete računar i pokrenuti GNU/Linux sa CD-a! Ovo vam potpuno omogućava da isprobate novi operativni sistem pre nego što se odlučite da ga instalirate na vašem računaru. Međutim, Ubuntu nije potpuno slobodan softver; sadrži vlasničke drajvere, firmware i aplikacije.
LibreOffice - Ovo je odličan, vođen i razvijen od strane zajednice, kancelarijski paket sa kojim dobijate, između ostalog, programe za pisanje, prezentacije, tabelarne proračune i crtanje. Sa njim čak možete sa lakoćom da otvorite i uređujete MS Word i MS PowerPoint fajlove. On radi na skoro svim platformama i potpuno je free, libre and open source software.
Mozilla Firefox - Ovo je *najbolji* veb pretraživač. On je neverovatno brz i dobija odlične kritike za svoje osetljive i impresivne karakteristike. Koncept ekstenzije omogućava Vam da koristite bilo kakav plugin (dodatak).
Njen prateći proizvod, Thunderbird je odličan klijent za elektronsku poštu, koji će da učini da čitate Vaše poruka brzinom puckanja prstima.
Mono - To je open source implementacija Microsoft .NET platforme. On omogućava .NET aplikacijama da budu kreirane i pokrenute na GNU/Linux, Windows, FreeBSD, Mac OS i mnogim drugim platformama.
Apache web server - Ovo je popularan open source Web server. U stvari, to je *najpopularniji* web server na planeti! On pokreće skoro više od polovine sajtova. Da, to je tačno - Apache "vozi" više sajtova od svih konkurenata (uključujući i Microsoft IIS) zajedno.
VLC Player - Ovo je video plejer koji može da reprodukuje bilo šta od DivX do MP3
pa Ogg preko VCDa i DVDa pa nadalje... ko kaže da open source nije zabavan? ;-)
Ovaj spisak je samo namenjen da vam dati kratak ideju - postoji mnogo više odličnih FLOSS, kao što su Perl jezik, PHP jezik, Drupal sistem za upravljanje sadržajem za sajtove, PostgreSQL server baze podataka, TORCS trkačka igra, KDevelop IDE, Xine - program za reprodukciju video sadržaja, VIM urednik, Quanta+ urednik, Banshee audio plejer, GIMP program za uređivanje slika ... Ovaj spisak može da ide u nedogled.
Da biste dobili najnovije vesti iz FLOSS sveta, pogledajte sledeće sajtove:

- linux.com
- LinuxToday
- NewsForge
- DistroWatch

Posetite sledeće sajtove za više informacija o FLOSS:

- GitHub Explore
- OMG! Ubuntu!
- SourceForge
- FreshMeat

Dakle, samo napred i istražujte ogroman, slobodan i otvoren svet FLOSS-a!

20 Colophon

Skoro sav softver koji sam koristio u stvaranju ove knjige je FLOSS.

20.1 Rađanje knjige

U prvom nacrtu ove knjige, ja sam koristio Red Hat 9.0 Linux, kao temelj mojih podešavanja, a sve do šestog nacrta, koristio sam Fedora Core 3 Linux kao osnovu mog rada.
Prvobitno sam koristio program KWord za pisanje knjige (kao što je objašnjeno u poglavlju "Lekcija iz istorije" u predgovoru).

20.2 Tinejdžerske godine

Kasnije sam prešao na DocBook XML koristeći Kate ali sam uvideo da je to suviše zamorno. Dakle, ja sam prebacio na OpenOffice koji je pružio odličan nivo kontrole za formatiranje kao i generisanje PDF-a, ali je proizvodio vrlo "prljav" HTML iz dokumenta.
Konačno sam otkrio XEmacs i prepisao sam knjigu od nule u DocBook XML (opet), nakon čega sam i odlučio da je ovaj format dugoročno rešenje.
U šestom izdanju, odlučio sam da upotrebljavam Quanta+ za uređivanje. Koristio sam standardni XSL stylesheets koji dolazi sa instalacijom Fedora Core 3 Linux-a. Kako god, napisao sam CSS dokument da daje boju i stil HTML stranicama. Takođe sam napisao grub leksički analizator, u Python-u naravno, koji mi je automatski obezbeđivao "bojenje" sintakse za sve programe iz knjige.
Za ovo sedmo izdanje, ja koristim MediaWiki kao osnovu mog podešavanja . Sada knjigu mogu uređivati direktno na mreži i svi čitaoci mogu direktno da čitaju/izmene/razgovaraju u okviru wiki sajta.
Koristio sam Vim za uređivanje zahvaljujući ViewSource ekstenziji za Firefox koja se integriše sa Vim-om.

20.3 Sada

Koristim Vim, Pandoc, i Mac OS X.

20.4 O autoru

http://www.swaroopch.com/about/

21 Istorija revizija

- 2.0
  - 20 Okt. 2012
  - Prepisana u Pandoc format , zahvaljujući mojoj ženi, koji je uradila većinu konverzije iz Mediawiki formata.
  - Pojednostavljivanje teksta, uklanjanje nebitnih delova kao što su nonlocal i metaklase
- 1.90
  - 04 Jul 2008, rad je još uvek u toku
  - Oživljavanje posle perioda od 3,5 godine!
  - Korekcija na Python 3.0
  - Prepisana koristeći MediaWiki (opet)
- 1.20
  - 13 Jan 2005
  - Kompletan prepis korišćenjem Quanta+ na Fedora Core 3 sa mnogo ispravki i dopuna. Mnogo novih primera. Prepisao sa DocBook od nule.
- 1.15
  - 28 Mar 2004
  - Manje revizije
- 1.12
  - 16 Mar 2004
  - Nešto je dodato, nešto je ispravljeno.
- 1.10
  - 09 mar 2004
  - Više korekcija grešaka u kucanju, zahvaljujući mnogim entuzijastima i čitaocima.
- 1.00
  - 08 Mar 2004
  - Nakon ogromne povratne informacije i sugestije od čitalaca, ja sam napravio značajne izmene u sadržaju uz korekcije grešaka.
- 0.99
  - 22 Jan 2004
  - Dodato je novo poglavlje o modulima. Dodati detalji o promenljivom broju argumenata u funkcijama.
- 0.98
  - 16 Jan 2004
  - Napisao Python skriptu i CSS stylesheet da poboljšam XHTML izlaz, uključujući grub-ali-funkcionalan leksički analizator za automatsko isticanje sintakse slično Vim-u prilikom listanja programa.
- 0.97
  - 13 Jan 2004
  - Još jedan potpuno prepisan nacrt, u DocBook XML (ponovo). Knjiga je poboljšana mnogo - mnogo je ujednačenija i čitljivija.
- 0.93
  - 25 Jan 2004
  - Dodato korišćenje IDLE-a i specifične stvari u Windows-u

- 0.92
  - 05 Jan 2004
  - Promene u nekoliko primera.
- 0.91
  - 30 dec 2003
  - Ispravljeno greške u kucanju. Poboljšane mnoge teme.
- 0.90
  - 18 dec 2003
  - Dodata još 2 poglavlja. OpenOffice format sa revizijama.
- 0.60
  - 21 novembar 2003
  - Potpuno prerađena i proširena.
- 0.20
  - 20 novembar 2003
  - Ispravljena neke greške u kucanju i greške u knjizi.
- 0.15
  - 20 novembar 2003
  - Pretvoreno u DocBook XML sa XEmacs.
- 0.10
  - 14 novembar 2003
  - Inicijalni nacrt korišćenjem Kword-a.

22 Prevodi

Postoje mnogo prevoda ove knjige, dostupne na različitim ljudskim jezicima, zahvaljujući mnoštvu neumornih volontera!
Ako želite da pomognete sa ovim prevodima, pogledajte listu volontera i jezika ispod i odlučite da li želite da započnete novi prevod ili da pružite pomoć u postojećim prevodilačkim projektima.
Ako planirate da započnete nov prevod, pročitajte kako da prevedete.

22.1 Arapski

Ispod je link za arapsku verziju. Zahvaljujući Ashraf Ali Khalaf na prevođenju knjige, možete da pročitate celu knjigu na internetu ovde , ili možete da je preuzmete sa sourceforge.net za više infromacija kliknite ovde.

22.2 Brazilski Portugalski

Postoje dva prevoda:
Samuel Dias Neto (samuel.arataca@gmail.com), napravio je prvi Brazilski Portugalski prevod ove knjige kada je Python bio u 2.3.5 verziji.
Samuelovo prevod je dostupan na aprendendopython.
Rodrigo Amaral (rodrigoamaral@gmail.com) je volonterski preveo knjigu na Brazilsko Portugalski.
Rodrigov je prevod dostupan na http://www.swaroopch.org/notes/Python_ptbr:Indice.

22.3 Katalonski

Moises Gomez (moisesgomezgiron@gmail.com) je volontirao da prevede knjigu na Katalonski. Prevod je u toku, a bio je prisutan u erstwhile wiki.
Moisès Gómez - Ja sam programer i nastavnik programiranja (obično za ljude bez ikakvog prethodnog iskustva).
Pre izvesnog vremena mi je bilo potrebno da naučim kako da programiram u Python-u, pa je Swaroop-ov rad zaista poslužio. Jasan, koncizan i sasvim dovoljan. Baš ono što mi je trebalo.
Posle ovakvog sopstvenog iskustva sa ovom knjigom, mislio sam da bi neki drugi ljudi iz moje zemlje mogli, takođe, da imaju neke koristi od ove knjige. Ali znanje Engleskog jezika može da bude prepreka.
Pa, zašto da ne pokušam da je prevedem? I to sam i uradio za prethodnu verziju knjige.
U mojoj zemlji postoje dva zvanična jezika. Ja sam izabrao Katalonski jezik pod pretpostavkom da će neko drugi prevesti na više rasprostranjen – Španski.

22.4 Kineski

Evo je kineska verzija na http://www.swaroopch.org/notes/Python_cn:Table_of_Contents.
Juan Shen (orion_val@163.com) je volonterski preveo knjigu na Kineski jezik. Ovaj prevod je dostupan na http://www.pycn.org/python%E5%9C%A8%E7%BA%BF%E6%89%8B%E5%86% .
Ja sam na postdiplomskim studijama na Wireless Telecommunication Graduate School,
Beijing University of Technology, China PR. Moje trenutno interesovanje je istraživanje sinhronizacije, channel estimation i multiuser detection of multicarrier CDMA sistema. Python je moj glavni programski jezik za svakodnevne simulacije i istraživački posao, uz pomoć Python Numeric-a. Python sam naučio samo pola godine ranije, ali kao što možete da vidite, on je zaista lako razumljiv, jednostavan za korišćenje i produktivan. Baš kao što je napisano u Swaroop-ovoj knjizi: "To je moj omiljeni programski jezik sada". "Zagrljaj Pitona" je bio moj tutorijal za svet

Python-a. Ova knjiga je jasan i efikasan put koji vas vodi u svet Python-a u najkraćem mogućem roku. Nije previše dugačka, ali efikasno pokriva skoro sve važne stvari u Python-u. Mislim da bi "Zagrljaj Pitona" trebalo da bude najbolja preporuka za početnike, kao njihov prvi Python tutorial. Zato namenjujem svoj prevod potencijalnim milionima Python korisnika u Kini.

22.5 Tradicionalni Kineski

Fred Lin (gasolin@gmail.com) je volontirao da prevede knjigu na Tradicionalni Kineski.
Ona je dostupna na http://code.google.com/p/zhpy/wiki/ByteOfZhpy.
Zanimljiva karakteristika ovog prevoda je da on takođe sadrži *izvornu izvršnu Python datoteku na Kineskom jeziku*, rame uz rame sa originalnim Python izvorom.
Fred Lin - Radim kao inženjer network firmware-a u Delta Network-u, a takođe sam saradnik TurboGears web framework-a.
Kao propovednik Python-a (:-p), trebalo mi je malo materijala za promociju Python jezika. Naleteo sam na "Zagrljaj Pitona" kao prelepu udarnu tačku kako za početnike, tako i za iskusne programere. "Zagrljaj Pitona" razrađuje osnove Python-a sa prihvatljivom dužinom.
Prevod je prvobitno zasnovan na pojednostavljenom Kineskom jeziku, i uskoro su urađene mnoge korekcije uz napor da se stane rame-uz-rame sa trenutnom wiki verzijom, kako bi se poboljšao kvalitet čitanja.
Najnovija verzija na Tradicionalnom Kineskom jeziku takođe sadrži i izvorni kod izvršnog Python-a na Kineskom jeziku, što je postignuto od strane mog novog 'zhpy' (Python na Kineskom) projekta (počeo Avgusta 2007-me).
zhpy (predstavlja (Z.H.?, ili zippy) gradi sloj za Python (layer) da prevodi ili vrši interakciju sa Python-om na Kineskom (Tradicionalnim ili Pojednostavljenim). Ovaj projekat je uglavnom namenjen za obrazovanje.

22.6 Francuski

Gregory (coulix@ozforces.com.au) je volonterski preveo knjigu na Francuski.
Gérard Labadie (Palmipede) je završio sa prevođenjem knjige na Francuski, ona počinje sa http://www.swaroopch.org/notes/Python_fr:Table_des_Mati%C3%A8res.

22.7 Nemački

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengeleinat@gmail.com) i Christoph Zwerschke (cito@online.de) su volonterski preveli knjigu na Nemački.
Njihov prevod se nalazi na http://abop-german.berlios.de.
Lutz Horn kaže:
Ja imam 32 godine i ima diplomu iz matematike sa University of Heidelberg, Germany. Trenutno radim kao softverski inženjer na javno-finansiranom projektu izgradnje web portala za sve stvari vezane za kompjuterske nauke u Nemačkoj. Glavni jezik koji koristim u svojoj profesiji je Java, ali, u pozadini, se trudim da uradim što je više moguće u Python-u. Posebno analiza i konverzija teksta je veoma laka u Python-u. Nisam baš upoznat sa GUI kompletima alata, jer je većina mojih programa su web aplikacije, gde se korisnički interfejs gradi korišćenjem Jave. Trenutno pokušavam da više koristim funkcionalno programiranje u Python-u i generatore. Posle kratkog korišćenja Rubi-ja, bio sam veoma impresioniran sa upotrebom blokova u ovom jeziku. Generalno volim dinamičnu prirodu jezika kao što su Pajton i Rubi, jer mi takva njihova priroda omogućava da radim stvari koje nisu moguće u mnogim statičkim jezicima, kao što je npr Java. Ja sam tražio neku vrstu uvoda u programiranje, koji odgovara učenju kompletnog nepoznavaoca programiranja. Našao

sam knjigu "Kako razmišljati kao kompjuterski naučnik: Učenje sa Python-om", i "Dive into Python". Prva je dobra za početnike, ali predugačka za prevod. Druga nije pogodna za početnike. Mislim da 'Zagrljaj Pitona' leži sasvim lepo između njih, jer nije predugačka, napisana je do savršenosti, a u isto vreme je dovoljno opširna da nauči početnike. Osim toga, ja volim jednostavnu DocBook strukturu, koji čini prevođenje teksta i generisanje izlaza u različitim formatima prelepom čarolijom.
Bernd Hengelein kaže:
Lutz i ja ćemo zajedno raditi na Nemačkom prevodu. Mi smo tek počeli sa uvodom i predgovorom, ali ćemo vas upoznati sa napretkom koji smo napravili. Ok, sada neke lične stvari o meni. Ja imam 34 godine i igram se računarima još od 1980-ih, kada je "Commodore C64" vladao svetom. Nakon studija računarskih nauka počeo sam da radim kao softverski inženjer. Trenutno radim u oblasti medicinskih snimanja za jednu veliku Nemačku kompaniju. Iako je C++ glavni jezik koji (moram) da upotrebljavam za moj svakodnevni rad, stalno sam u potrazi za novim stvarima za učenje. Prošle godine sam se zaljubio u Python, jer je divan jezik, zbog svojih mogućnosti i lepote. Pročitao sam negde na internetu o momku koji je rekao da voli Python, jer kod izgleda tako lepo. Po mom mišljenju, on je apsolutno u pravu. Tada sam odlučio da naučim Python, pa sam primetio da postoji veoma malo dokumentacije na Nemačkom jeziku, koja je na raspolaganju. Kada sam došao do ove knjige, pala mi je na pamet spontana ideja o Nemačkom prevodu. Srećom, Lutz je imao istu ideju i sada možemo da podelimo posao. Takođe se radujem dobroj saradnji!

22.8 Grčki

Grčka Ubuntu zajednica je prevela ovu knjigu na Grčki, za korišćenje u njenim on-line asinhronim Python lekcijama koje se odvijaju na njenim forumima. Kontaktirajte @savvasradevic za više informacija.

22.9 Indonežanski

Daniel (daniel.mirror@gmail.com) je preveo knjigu na indonežanski na http://python.or.id/moin.cgi/ByteofPython.
W. Priyambodo je takođe volontirao u prevodu knjige na Indonežanski. Prevod je u toku, a nalazi se ovde http://www.swaroopch.org/notes/Python_id:Daftar_Isi.

22.10 Italijanski

Enrico Morelli (mr.mlucci@gmail.com) i Massimo Lucci (morelli@cerm.unifi.it) su se dobrovoljno prijavili da prevedu knjigu na Italijanski jezik.
Italijanski prevod je prisutan na www.gentoo.it/Programmazione/byteofpython.
Novi prevod je u toku i počinje na http://www.swaroopch.org/notes/Python_it:Prefazione.
Massimo Lucci i Enrico Morelli - radimo na University of Florence (Italy) - Odeljenje Hemije. Ja (Massimo) radim kao servis inženjer i sistem administrator na spektometru nuklearne magnetne rezonance, Enriko radi kao servis inženjer i sistem administrator za naš CED i paralelne/klaster sisteme. Mi programiramo u Python-u već oko sedam godina, mi imamo iskustvo u radu na Linux platformama oko deset godina. U Italiji smo odgovorni i administratori za www.gentoo.it, web sajt Gentoo/Linux distrubucije i www.nmr.it (trenutno u izgradnji) koji se bavi stvarima oko nuklearne magnetne rezonance i aplikacije Kongresne Organizacija i menadžmenta. To je sve! Impresionirani smo mudrim jezikom kojim se koristi vaša knjiga i mislimo da je to neophodno za približavanje Python-a novim korisnicima (mislimo o stotinama studenata i istraživača koji rade u našim laboratorijama).

22.11 Japanski

Evo je Japanska verzija na http://www.swaroopch.org/notes/Python_ja:Table_of_Contents.
Shunro Dozono (dozono@gmail.com) je preveo knjigu na Japanski.

22.12 Mongolski

Ariunsanaa Tunjin (luftballons2010@gmail.com) je volonter u prevodu knjige na Mongolski.
Novost za Novembar 22, 2009: Ariunsanaa je skoro završila prevod.

22.13 Norveški (bokmål)

Eirik Vågeskar (http://www.swaroopch.org/notes/User:Vages) je srednjoškolac Sandvika videregående skole u Norveškoj, bloger i trenutno prevodi knjigu na Norveški (bokmål). Prevod je u toku, a vi možete da ga vidite na http://www.swaroopch.org/notes/Python_nbno:Innholdsfortegnelse za više detalja.
Eirik Vågeskar: Uvek sam želeo da programiram, ali zato što govorim veoma malo jezika, proces učenja je bio mnogo teži. Većina tutorijala i knjiga su napisani na veoma tehničkom Engleskom, tako da većina maturanata neće ni imati potreban vokabular da shvate šta piše u tutorijalima. Kada sam otkrio ovu knjigu, svi moji problemi su bili rešeni. "Zagrljaj Pitona" koristi jednostavan netehnički jezik da bi objasnila programski jezik koji je isto tako jednostavan, a sa ove dve stvari učenje Python-a je zabavno. Nakon čitanja polovine knjige, ja sam odlučio da je knjiga vredna prevođenja. Nadam se da će prevod pomoći ljudima koji se nalaze u istoj situaciji kao i ja (naročito mladi ljudi), a možda i pomogne širenju interesovanja za ovaj jezik među ljudima koji imaju manje tehničko znanje.

22.14 Poljski

Dominik Kozaczko (dkozaczko@gmail.com) je volontirao da prevede knjigu na Poljski. Prevod je u toku i njegova glavna internet strana je dostupna ovde: Ukas Pythona.
Novija informacija: prevod je kompletan i spreman od Oktobra 2 2009g. Zahvaljujem Dominiku, njegova dva učenika i njihovim prijateljima na uloženom vremenu i trudu!
Dominik Kozaczko - Ja sam učitelj kompjuterskih nauka i informacionih tehnologija.

22.15 Portugalski

Fidel Viegas (fidel.viegas@gmail.com) je volonterski preveo knjigu na Portugalski.

22.16 Rumunski

Paul-Sebastian Manole (brokenthorn@gmail.com) je volontirao da prevede ovu knjigu na Rumunski.
Paul-Sebastian Manole - Ja sam student računarstva druge godine u Spiru Haret University u Rumuniji. Ja sam samouki programer i odlučio sam da naučim novi jezik - Python. Na Web-u mi je rečeno da ne postoji bolji način da se to uradi od čitanja "Zagrljaj Pitona". Eto koliko je popularna ova knjiga (čestitke autoru za pisanje takvog dela koje je lako za čitanje). Počinje da mi se sviđa Python pa sam odlučio da pomognem prevesti najnoviju verziju Swaroop-ove knjige na Rumunski. Iako sam ja inicijator, ja sam samo jedan volonter, pa ako možete da pomognete, molim vas da mi se pridružite.
Prevod se radi na http://www.swaroopch.org/notes/Python_ro.

22.17 Ruski i Ukrajinski

Averkiev Andrey (averkiyev@ukr.net) je volonter da prevede knjigu na Ruski, a možda i Ukrajinski (Ukoliko to vreme dozvoli).
Vladimir Smolyar (v_2e@ukr.net) je počeo Ruski prevod wiki stranice. Možete pročitati razvojnu verziju na http://www.swaroopch.org/notes/Python_ru:Table_of_Contents .

22.18 Slovački

Albertio Ward (albertioward@gmail.com) je preveo knjigu na Slovački fatcow.com/edu/python-swaroopch-sl/ :
Mi smo neprofitna organizacija pod nazivom "Prevod za obrazovanje". Mi predstavljamo grupu ljudi, uglavnom studenata i profesora, sa Slavonic University. Sačinjeni smo od studenata iz različitih odeljenja: lingvistike, hemije, biologije, itd . Mi pokušamo da pronađemo interesantne publikacije na internetu koje bi mogle biti relevantne za nas i naše univerzitetske kolege. Ponekad nađemo članke sami; neki put naši profesori nam pruže pomoć da nađemo materijal za prevod. Nakon dobijanja dozvole od autora, prevodimo tekstove i postavimo ih na našem blogu koji je dostupan i pristupačan našim kolegama i prijateljima. Ove prevedene publikacije često pomažu studentima u njihovom svakodnevnom učenju.
Zašto sam izabrao ovu knjigu za prevod? Prevod je napravljen da pomogne Bugarima (prevodilac na srpski: ???) da razumeju ovu knjigu do najmanjih mogućih detalja. Pošto sam uradio neka istraživanje vezano za popularnost i relevantnost teme, shvatio sam da je zaista hitno za sve one koji žive u mojoj zemlji. Dakle, mislim da može da postane veoma popularna. Jezička barijera u ovom malom slučaju više ne postoji, jer je eliminisana mojim prevodom.

22.19 Španski

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) i Cristian Bermudez Serna (crisbermud@hotmail.com) su se dobrovoljno prijavili da prevedu knjigu na Španski. Prevod je u toku, možete pročitati Španski (Argentinski) prevod na http://www.swaroopch.org/notes/Python_es-ar:Tabla_de_Contenidos .
Gustavo Echeverria kaže:
Radim kao softverski inženjer u Argentini. Ja uglavnom koristim C# i .NET tehnologiju na poslu ali obavezno Python ili Ruby u mojim ličnim projektima. Čuo sam, pre mnogo godina, za Python i u njemu sam se momentalno zaglavio. Ne tako dugo posle otkrivanja Python-a, otkrio sam i ovu knjigu, i ona mi je pomogao da naučim taj jezik. Onda sam se dobrovoljno odlučio da prevedem knjigu na Španski. Sada, nakon dobijanja nekih zahteva, ja sam počeo sa prevođenjem knjige "Zagrljaj Pitona" uz pomoć Maximiliano Soler.
Cristian Bermudez Serna kaže:
Ja sam student telekomunikacijskog inženjeringa na University of Antioquia (Colombia). Pre više meseci, sam počeo da uči Pajton i naišao sam na ovu divnu knjigu, pa sam se dobrovoljno javio da uradim Španski prevod.

22.20 Švedski

Mikael Jacobsson (leochingkwake@gmail.com) je dobrovoljac u prevodu knjige na Švedski.

22.21 Turski

Türker SEZER (tsezer@btturk.net) i Bugra Cakir (bugracakir@gmail.com) su se dobrovoljno prijavili da prevedu knjigu na Turski. Gde je Turska verzija? Bitse de okusak.

23 Kako prevoditi

Pun izvor knjige je dostupan iz GIT-spremišta https://github.com/swaroopch/byte_of_python .
Molimo forkujte repozitorijum.
Zatim, uskladištite repozitorijum na računaru. Vi treba da znate kako da koristite GIT da bi se to uradilo.
Počnite uređivanje .pd datoteke na Vašem lokalnom jeziku. Molimo vas da pročitate Pandoc README da bi razumeli formatiranje teksta.
Zatim sledite uputstva napisana u README da instalirate softver koji je potreban da bi mogli da konvertujete sirove izvorne datoteke u PDF format, itd.