Poglavlje 1.

Uvod

Uslijed velikog broja zabavnih, uzbudljivih i praktičnih primjena, elektronička računala su postala nezaobilazni dio naše svakidašnjice. Primjeri takvih primjena su rukovanje digitalnim sadržajima (glazbom, fotografijama, videom), pregledavanje i pretraživanje mrežnih stranica, razonoda u virtualnim svjetovima, provođenje novčanih transakcija (kupovina s dostavom, kućno bankarstvo), satelitska navigacija, vremenska prognoza, te upravljanje strojevima i robotima. Međutim, i najsofisticiranija današnja računala sastavljena su od prilično primitivnih elektroničkih sklopova koji su sposobni izvršavati samo jednostavne operacije čija složenost nije puno veća od korjenovanja ili dijeljenja decimalnih brojeva. Da bismo ostvarili inteligentno ponašanje, računalu trebamo udahnuti život organiziranjem primitivnih sklopovskih operacija u računske postupke (engl. *computational process*). Računski postupci su apstraktna "bića" koja ne možemo ni vidjeti ni opipati, svojevrsni duhovi koji nastanjuju (opsjedaju?) računala te određuju njihovo ponašanje i utjecaj na fizički svijet. Budući da računske postupke izražavamo (ili zadajemo) programima (engl. *computer program*), programiranje je temeljna vještina entuzijasta i profesionalaca koji projektiraju i izrađuju primjene temeljene na računalima.

1.1. O programiranju

U svakodnevnom govoru pod programiranjem podrazumijevamo oblikovanje, pisanje, ispitivanje i održavanje računalnih programa odnosno programske potpore. Svaki računalni program možemo promatrati kao formalni dokument koji izražava napredovanje računskog postupka prikladnog za automatsku obradu podataka. Podatci koje obrađujemo mogu biti unaprijed poznati (kao npr. kod programa za komprimiranje podataka), pribavljeni u stvarnom vremenu (npr. kod programa za očitavanje linijskog kôda), ili zadani interaktivno od strane korisnika (npr. kod programa za izradu tehničkih crteža). Tipični današnji računalni programi su vrlo složeni te nastaju nakon vi-

šegodišnjih napora programskih inženjera organiziranih u razvojne timove. Međutim, pokazat ćemo kako svi programi nastaju iz jednostavnih i lako razumljivih građevnih jedinica, baš kao što i naše najsloženije misli uspijevamo izraziti kombiniranjem pojmova iz ograničenog rječnika prirodnog jezika.

Tehnički gledano, konačna svrha svakog programa je da njegovo strojno tumačenje pobudi željeni računski postupak. Ali, programi trebaju biti razumljivi i ljudima, kako samim autorima tako i njihovim kolegama. U praksi, svaki iole korišteni program treba neprestano održavati: ispravljati nedostatke, prilagođavati ga novim zahtjevima te unositi razne druge vrste poboljšanja. Stoga programeri puno više vremena provode u čitanju postojećih programa nego pri pisanju novih. Tako dolazimo do možda iznenađujuće spoznaje da će pri razvijanju programa vrijediti slične stilske odrednice kao i pri izražavanju u prirodnom jeziku. U načelu, više ćemo cijeniti jednostavna i sažeta rješenja od opširnih i kompliciranih, nemilosrdno ćemo izbacivati nepotrebne i zastarjele dijelove kôda, a od ponavljanja ćemo bježati kao vrag od tamjana. Veće programe ćemo razdjeljivati u "odjeljke" posvećene pojedinim važnim temama na prikladnoj razini detalja, u skladu s organizacijskim kriterijima koje proučava disciplina programskog oblikovanja (engl. *software design*).

Vidimo da pri izražavanju računskih postupaka programima trebamo pomiriti dva važna kriterija. Programi moraju istovremeno biti i intuitivni (kako bi bili jasni ljudima) i formalno egzaktni (kako bi tumačenje bilo nedvosmisleno i efikasno). Danas dominantni pristup za ispunjavanje tih zahtjeva je zadavanje programa u obliku tekstovnog izvornog kôda (engl. *source code*), u skladu s pravilima odabranog programskog jezika. Izvorni kôd izražava računski postupak odnosno definira funkcionalnost programa u konkretnoj primjeni, te stoga predstavlja središnju točku u našem pogledu na programiranje.

1.2. Programski jezici

Programiranje je vrlo dinamična tehnička disciplina, u okviru koje se neprestano događaju preslagivanja i izmjene. Prvi razlog za dinamiku područja je njegova relativna mladost. Iako se ideje automatske obrade podataka javljaju već pri kraju 19. stoljeća u radovima Babbagea, Lovelace i Holleritha, područje doživljava procvat tek u drugoj polovici 20. stoljeća, pojavom poluvodičke tehnologije, digitalnih računala i viših programskih jezika opće namjene. Viši programski jezici koriste matematičke simbole i riječi iz engleskog jezika za zadavanje operacija. Zato je pisanje programa u njima znatno jednostavnije nego pisanje strojnih instrukcija, a napisani programski kôd je puno čitljiviji i pregledniji. Programski jezici opće namjene koncipirani su tako da se mogu koristiti u raznolikim područjima primjene, za razliku od programskih jezika specifične namjene poput SQL-a koji je namijenjen isključivo za rukovanje podatcima u bazama podataka ili HTML-a koji služi za stvaranje mrežnih stranica. Prvi viši pro-

gramski jezik Fortran razvijen je prije manje od 6 desetljeća s namjenom da olakša pisanje programa za numeričke proračune te se i danas koristi i razvija (posljednja inačica definirana je standardom ISO/IEC 1539-1:2010 iz 2010. godine). Fortran je začetnik kategorije prevođenih jezika (engl. *compiled languages*) koji su koncipirani s ciljem beskompromisnog ostvarivanja što veće brzine izvođenja i što manjih memorijskih zahtjeva programa. Moderniji predstavnici te kategorije su jezici Pascal, C, Ada, C++, itd.

Drugi razlog za dinamičnost područja je stalni porast procesorske moći i memorijskog kapaciteta računala. U odnosu na 1978. godinu kad se pojavio VAX 11/780 kao jedno od prvih "pravih" računala s 32-bitnim virtualnim adresama, današnja računala opće namjene imaju oko 10 000 puta veću procesnu moć te oko 50 000 puta prostraniju memoriju. Međutim danas smo svjedoci i novog višejezgrenog trenda koji bi u nadolazećem desetljeću mogao dovesti do nove, "mnogojezgrene" revolucije u računarstvu. Stalni dotok nove procesne moći doveo je do situacije u kojoj procesorsko vrijeme prestaje biti kritičnim resursom u mnogim domenama primjene. Posljedično, krajem 1980-ih i početkom 1990-ih godina novi jezici su nicali poput gljiva poslije kiše: Perl, TCL, Visual Basic, Python, Ruby, odnosno Java i C♯. Ovdje valja spomenuti da su inovacije na kojima se ovi jezici temelje ipak razvijene znatno ranije u okviru jezika kao što su Smalltalk (1972.), CLU (1974.) i Lisp (1958.). Glavne od tih inovacija su objektno orijentirano programiranje, automatsko rukovanje memorijom, dinamičko prevođenje, implicitno i dinamičko tipiziranje, te integrirana razvojna okolina. Zajednička ambicija svih tih jezika bila je upregnuti nagomilani višak performanse u cilju ubrzavanja razvojnog procesa te ostvarivanja bolje jasnoće, sigurnosti i platformske neovisnosti izvornog kôda.

Jedna od važnih značajki jezika novog vala bilo je odustajanje od izravnog prevođenja izvornog kôda u strojni kôd, što je jedna od osnovnih pretpostavki prevođenih jezika poput Fortrana. Tako se programi novijih jezika tipično isporučuju ili u originalnom izvornom kôdu (Perl, TCL, Visual Basic, Python, Ruby) ili u kompaktnom međukôdu (Java, C♯, Python), te interpretiraju na programski izvedenom računalu koje ćemo nazivati virtualnim strojem (engl. *virtual machine*). Nadalje, većina novijih jezika koristi se i automatskim upravljanjem memorijom. Glavne prednosti ovih dviju inovacija su veća platformska neovisnost te veća sigurnost uslijed bržeg i lakšeg detektiranja pogrešaka. Pokazuje se da te inovacije ne moraju imati velike posljedice na brzinu izvođenja programa, pod uvjetom da računalo može tolerirati višestruko veće memorijske zahtjeve. Jedno istraživanje relativne performanse[1] pokazuje da je u prosjeku Java samo dvostruko sporija od C++-a uz oko sedam puta veće zauzeće memorije. Međutim, omjeri brzina izvođenja jako variraju za pojedinačne probleme. Kod polovice primjera programi u Javi su bili više od dva puta sporiji od odgovarajućih programa pisanih u jeziku C++. Razlog tome su sofisticirani virtualni strojevi koji izvorni kôd (ili međukôd) mogu prevesti u optimalni strojni kôd tijekom samog izvođenja programa.

[1] `http://benchmarksgame.alioth.debian.org/`

Ovu tehniku nazivamo dinamičkim prevođenjem (engl. *just in time compilation*, JIT).

Većina novijih jezika (TCL, Perl, Python, Ruby) dodatno se koristi i idejom dinamičkog tipiziranja (engl. *dynamic typing*)[2]. U takvim jezicima simbolička imena podataka u programu (kolokvijalno: varijable) nisu nerazdvojno povezana s jedinstvenim tipom te s jedinstvenom memorijskom lokacijom na kojoj je pohranjen pridruženi podatak. Python i Ruby kao po mnogome najinovativniji jezici odlaze i korak dalje od toga pa tipiziranje simboličkih imena istovremeno provode i dinamički, i implicitno (engl. *duck typing*) i strogo (engl. *strong typing*). Ove pojmove ćemo detaljno obraditi u tekstu knjige, a ovdje ćemo samo reći da ta svojstva čine Python i Ruby iznimno izražajnim i jezgrovitim, ali i, za sada, ne toliko efikasnim jezicima opće namjene. Naime, usporedba performansi[3] pokazuje da su programi pisani u Pythonu oko dvostruko sažetiji od istovjetnih programa u Javi, ali istovremeno i oko 14 puta sporiji. S druge strane i Java je sve donedavno imala vrlo loše performanse, ali je napretkom na području tehnika dinamičkog prevođenja uspjela zauzeti uvažavajuće pozicije u odnosu na prevođene jezike. Očekujemo da bi se sličan scenarij relativnog poboljšanja performanse mogao dogoditi i s Pythonom, iako nije realno očekivati da bi Python u skoroj budućnosti mogao postati brži od Jave, a još manje od C++-a. Međutim, danas u mnogim domenama primjene efikasnost programa nije presudna, dok elegancija izvornog kôda te, posljedično, brzina razvoja, često čine glavnu razliku između uspješnih i neuspješnih programskih projekata. To je omogućilo Pythonu da doživi značajni porast popularnosti uslijed koje je u samo nekoliko godina prešao put od opskurnog projekta za koji je znao samo njegov tvorac pa sve do postavljanja novih standarda na području oblikovanja programskih jezika [4].

1.3. Programski jezik Python

Koncepcija programskog jezika Python oživotvoruje jedan od najavangardnijih pogleda na programiranje u trenutku pisanja ove knjige. Kao glavni kriteriji oblikovanja tog jezika odabrani su izražajnost i jasnoća izvornog koda. Stoga je izvorni kôd u Pythonu tipično i sažetiji i čitkiji nego što bi to bio slučaj kod ostalih popularnih jezika opće namjene (C, C++, Java, C♯), i to ne samo za programere s crnim pojasom. Nadalje, implementacije programskih komponenata u Pythonu obično pružaju neusporedivo veći potencijal ponovnog korištenja od odgovarajućih implementacija u ostalim jezicima. Tako je u Pythonu veza između objekata u memoriji računala i njihovih

[2] "Dinamičko" označava da se odgovarajuća aktivnost odvija tijekom izvođenja programa. Naprotiv, "statičke" aktivnosti se odvijaju prije izvođenja programa, tipično tijekom "klasičnog" prevođenja izvornog kôda u strojni kôd.

[3] `http://benchmarksgame.alioth.debian.org/`

[4] Prema indeksu popularnosti programskih jezika TIOBE (`http://www.tiobe.com/index.php/content/paperinfo/tpci/`), u listopadu 2015. godine Pythonu je pripadao udio od oko 5%. Za usporedbu, C i C++ zajedno su imali 22%, Java oko 20%, dok su svi ostali jezici bili ispod 5%.

simboličkih imena u izvornom kôdu programa znatno labavija nego kod C-a ili Jave. Tu razliku pokušat ćemo i terminološki naglasiti pa ćemo simbolička imena objekata umjesto varijablama radije zvati referencama ili naprosto imenima.

Po svojoj koncepciji, Python je dosljedan objektno orijentirani jezik u kojem se razredi mogu tretirati poput objekata, baš kao i u Smalltalku. Međutim, Python pored toga pragmatično podržava i ostale stilove programiranja. U posljednje vrijeme posebno postaje zanimljiv funkcijski stil koji je u Pythonu znatno bolje podržan nego u ostalim objektno orijentiranim jezicima. Tako su i funkcije u Pythonu objekti prvog reda, što znači da se mogu pohranjivati u podatkovnim strukturama, stvarati tijekom izvođenja programa, te prenositi kao povratna vrijednost potprograma.

Python je platformski neovisan jezik: većina programa može se izvoditi na Windowsima, na Linuxu, na Mac OS X-u, kao i na brojnim manje popularnim platformama. U praksi, ovo je prvenstveno moguće zbog vrlo opsežne standardne biblioteke koja pruža platformski specifičnu funkcionalnost iza transparentnog platformski neovisnog sučelja. Standardna biblioteka omogućava obavljanje složenih zadataka poput preuzimanja mrežne stranice ili komprimiranja i enkripcije podataka, i sve to u nekoliko redaka izvornog kôda. Dodatno, postoje tisuće nezavisnih biblioteka, koje u odnosu na standardnu biblioteku nude specijalizirane mogućnosti poput matričnih operacija ili detekcije lica u slikama[5].

Python je jezik s iznimno širokim područjem primjene, koji se u posljednje vrijeme nameće kao prvi izbor kad god procesorsko vrijeme nije kritično (kad bismo možda radije koristili C++), te kad grafičko korisničko sučelje nije složenije od glavne funkcionalnosti programa (kad bismo možda radije koristili Javu ili C♯). U usporedbi s ostalim jezicima visoke razine (npr. Perlom), za Python se često kolokvijalno kaže da znatno bolje skalira. Pojednostavljenim jezikom rečeno, u Pythonu se lijepo pišu i skripte od jednog ekrana i složeni programski sustavi sastavljeni od preko tisuću modula izvornog kôda.

Konačno, ovdje trebamo reći da se od prosinca 2008. usporedno razvijaju dvije stabilne varijante jezika, pri čemu novija varijanta Python 3.x nije u potpunosti kompatibilna sa starijom varijantom Python 2.x. Međutim, za krajnjeg korisnika to nije veliki problem, jer se dvije varijante uistinu razlikuju u nijansama, dok se standardna biblioteka i većina vanjskih biblioteka već duže vrijeme nude u dva paralelna izdanja. U svakom slučaju, nova varijanta jezika je vrlo dobro prihvaćena te će vjerojatno u doglednoj budućnosti u potpunosti potisnuti staru. Stoga smo odlučili da za ovu knjigu koristimo noviju varijantu jezika, odnosno Python 3.x[6].

[5] Važnije nezavisne biblioteke predstavljene su na mrežnim stranicama `http://pypi.python.org`.

[6] U vrijeme dovršavanja ove knjige najnovija je bila verzija 3.4.3

1.4. Pokretanje programa u Pythonu

Kao što smo već rekli, programi u Pythonu pokreću se u dva koraka. Izvorni kôd se prvo prevodi u kompaktni *međukod* (engl. *intermediate bytecode*), koji je isti bez obzira na kojoj će se platformi program u konačnici izvršavati. Prilikom pokretanja programa na odredišnom računalu taj se međukod interpretira na virtualnom stroju specifičnom za operacijski sustav dotičnog računala. Dakle, da bismo pokrenuli izvođenje programa u Pythonu, potrebno je na računalu imati instalirano odgovarajuće izvršno okruženje (engl. *runtime environment*), kao što je to slučaj i s jezicima Java ili C♯.

1.4.1. Instaliranje izvršnog okruženja

Izvršno okruženje Pythona vrlo je lako instalirati na svim popularnim i većini manje popularnih platformi. Za platforme MS Windows i Mac OS X preporučamo preuzeti odgovarajući instalacijski paket za Python 3.x sa službenih mrežnih stranica (`http://www.python.org/download`) te pokrenuti njegovo instaliranje dvostrukim klikom. Na većini popularnih distribucija UNIX-a Python će tipično biti predinstaliran. Ako to ipak nije slučaj s vašim sustavom, preporučamo koristiti se programom za preuzimanje i instaliranje paketa na kojem se temelji vaša distribucija. Npr, na umreženom računalu pod Debianom, Python i razvojno okruženje IDLE instalirali bismo naredbom:

```
$ sudo apt-get install python3.2 idle-python3.2
```

1.4.2. Interaktivni rad s Pythonom

Python nudi mogućnost interaktivnog zadavanja naredbi i programa, u skladu s uvriježenom tradicijom interpretiranih programskih jezika[7]. Do standardnog interaktivnog sučelja Pythona (u daljnjem tekstu: ljuske) dolazimo pokretanjem izvršnog programa `python3`[8]. Na Windowsima to možemo postići odabirom [Start, Run] te zadavanjem naredbe: `python3`, ili odabirom [Start, Programs, Python 3.4, Python (*command line*)]. Pythonova ljuska omogućava zadavanje upita (izraza ili naredbi) kojima možemo ispitivati postojeći izvorni kôd ili eksperimentirati s bibliotekama s kojima nemamo dovoljno prethodnog iskustva. Jednu takvu epizodu opisat ćemo u sljedećem primjeru.

[7] U engleskoj literaturi se ovakva interaktivna okruženja često označavaju kraticom REPL (engl. *read-eval-print-loop*).

[8] Postoje i druge ljuske koje nude poboljšanu funkcionalnost u odnosu na standardnu. Jedna od najzanimljivijih alternativnih ljuski je `ipython`.

Pretpostavimo da želimo iskušati rad sa znakovnim nizovima koje u Pythonu predstavljamo ugrađenim tipom `str`. Zadajmo znakovni niz `doručak`, te tražimo od Pythona da mu ispiše sadržaj:

```
>>> doručak = 'burek'
>>> doručak
'burek'
```

U prikazanom primjeru znakovi >>> predstavljaju odziv ljuske. Odziv ispisuje ljuska, dok tekst nakon odziva (`doručak = 'burek'`) odgovara upitu koji upisujemo. Ispisom odziva Python poručuje da je spreman za prihvat sljedećeg upita. U prvom retku primjera zadali smo upit kojim od Pythona tražimo da stvori znakovni niz `'burek'` te da mu dodijeli ime `doručak`. Upit iz drugog retka zahtijeva od Pythona da ispiše sadržaj imena `doručak`. Treći redak primjera sadrži rezultat izvršavanja upita iz drugog retka. Lako je pogoditi da taj tekst ispisuje Python jer na početku nema odziva >>>.

Sada pretpostavimo da se izravno iz interaktivne ljuske Pythona želimo upoznati s metodom `replace` ugrađenog tipa `str`. U sljedećem primjeru prvo tražimo ispis dokumentacije za tu metodu, a zatim primjenjujemo metodu prema dobivenim uputama:

```
>>> help(str.replace)
Help on method_descriptor:

replace(...)
    S.replace (old, new[, count]) -> str

    Return a copy of S with all occurrences of substring
    old replaced by new.  If the optional argument count is
    given, only the first count occurrences are replaced.
>>> doručak.replace('rek','htla')
'buhtla'
```

Skrećemo pažnju da smo i upitom `help(doručak.replace)` mogli doći do dokumentacije te da je za izlazak iz preglednika dokumentacije potrebno pritisnuti tipku `q`. Povijest zadanih naredbi možemo pregledavati tipkama sa strelicom prema gore odnosno dolje, kao što je slučaj i kod ostalih interaktivnih sučelja.

Na kraju bismo naglasili da interaktivni način pruža jednake mogućnosti kao i klasično neinteraktivno pokretanje izvornog kôda pohranjenog u datoteci. Dakle, možemo definirati funkcije i razrede, otvarati datoteke, zadavati petlje. Ukratko, bilo koji valjani konstrukt u Pythonu ispravno će se interpretirati i u interaktivnom načinu rada. Tu mogućnost ćemo vrlo često koristiti u tekstu knjige jer vjerujemo da je interaktivno eksperimentiranje najbolji način učenja programiranja.

1.4.3. Integrirana razvojna okolina IDLE

Programe u Pythonu možemo pokretati i iz integrirane razvojne okoline IDLE. IDLE nije dovršen proizvod, ali i u postojećem stanju nudi mogućnosti uređivanja izvornog kôda, kontroliranog izvođenja (debagiranja), te interaktivnog rada kao što je prikazano u prethodnom odjeljku (§1.4.2.). Budući da je pisan u Pythonu uz pomoć modula tkinter, IDLE radi na većini postojećih platformi.

IDLE možemo pokrenuti iz menija grafičkog korisničkog sučelja. Na Windowsima tipično odabiremo [Start, Programs, Python 3.4, IDLE], dok na Linuxu pod Gnomeom to postižemo s [Main Menu, Programming, IDLE 3]. Odmah po pozivanju otvara se interaktivno sučelje prema Pythonovu izvršnom okruženju. Nakon odabira (File, New) otvara se novi prozor za neinteraktivni unos novog modula izvornog kôda. Sada možemo utipkati naš prvi program:

```
print("Python ima kvačice u serijskoj opremi: šđžčćŠĐŽČĆ!\n")
```

i pohraniti ga odabirom (File, Save). Izvođenje programa pokrenut ćemo s (Run, Run Module), pri čemu se ispis programa prikazuje u glavnom prozoru IDLE-a.

IDLE nudi osnovne mogućnosti editiranja na koje smo se navikli u boljim editorima opće namjene: bojenje sintakse, automatsko uvlačenje, te automatsko dopunjavanje riječi (kombinacijama tipki `Alt+/` ili `Ctrl+<space>`). Pokretanje izvođenja modula u tekućem prozoru za editiranje postižemo s (Run, Run module). Konačno, postoje i osnovne mogućnosti debagiranja koje aktiviramo odabirom (Debug, Debugger).

Kao što smo već naveli, u trenutku pisanja ove knjige IDLE nije bio ispoliran proizvod (za razliku od Pythona kao jezika, njegove standardne biblioteke i pripadnog izvršnog okruženja). Primjerice, ispisivanje dokumentacije u prozoru s izvornim kôdom funkcionira samo ako se kursor nalazi nakon otvorene zagrade, iako dokumentacija postoji i za objekte koji se ne mogu pozivati (npr. za module). Debbuger je funkcionalan, ali zahtijeva više manualnog rada nego što smo to navikli u ostalim programima takve vrste (npr. treba odabrati dva menija za pokretanje debagiranja). Povijest zadanih naredbi ovdje se pregledava tipkama `Alt+p` odnosno `Alt+n`, što je puno manje intuitivno od tipki sa strelicama. Međutim, iako nije dovršen proizvod, IDLE jest funkcionalan, a vjerujemo da će s vremenom postajati sve bolji i bolji. Stoga ga preporučamo svima koji nisu pretjerano osjetljivi na vizualnu dopadljivost i intuitivnost sučelja.

1.4.4. Neinteraktivno pokretanje programa

Konačno, programe u Pythonu možemo pokretati i iz naredbenog sučelja (engl. *command line interface*) operacijskog sustava. Glavna prednost ovakvog pristupa je mogućnost pozivanja pythonskog kôda iz drugih programa (npr. iz automatiziranih administracijskih skripti) odnosno preko prečaca (engl. *shortcut*) na radnoj površini (engl. *desktop*) grafičkog korisničkog sučelja.

Za razliku od Jave, Python ne iziskuje prethodno prevođenje izvornog kôda u međukôd. To znači da se istim izvršnim programom (`python3`) možemo koristiti kako za pokretanje već prevedenog međukôda, tako i za istovremeno prevođenje i pokretanje datoteka s izvornim kôdom. Pretpostavimo da smo naš drugi program u Pythonu zadali tekstovnom datotekom 'drugi.py', čiji sadržaj možemo zadati i pohraniti iz interaktivnog sučelja IDLE:[9]

```
# ljestve označavaju komentar
# ovo je sadržaj datoteke 'drugi.py'
brojčina = 10**100 + 1
print("Python ide dalje od Googola: {}!\n".format(brojčina))
```

Sada program možemo istovremeno i prevesti i pokrenuti naredbom:

```
$ python drugi.py
```

Napominjemo da znak `$` u gornjem primjeru označava da ostatak retka predstavlja naredbu koju valja zadati preko naredbenog sučelja operacijskog sustava. Na Windowsima, naredbeno sučelje pruža program `Command Prompt`[10], do kojega se dolazi odabirom [Start, Programs, Accessories, Command prompt]. Na grafičkom sučelju Gnome (Linux odnosno UNIX) do naredbenog sučelja dolazimo odabirom [Main Menu, Accessories, Terminal]. Naša naredba navodi da želimo pozvati izvršni program `python3` s argumentom `drugi.py`. Python će pročitati datoteku zadanu argumentom te će njenom analizom zaključiti da se radi o izvornom kôdu koji treba prevesti u međukôd. Međukod će se konačno izvršiti, što će rezultirati ispisivanjem sljedeće poruke u prozoru naredbenog sučelja (znakovi sa strelicama označavaju da se ispis nastavlja u sljedećem retku):

```
Python ide dalje od Googola: 10000000000000000000000000000000000000000000000000 ↙
    ↳ 00000000000000000000000000000000000000000000000000000000000000000000000000 ↙
    ↳ 000001!
```

Ovdje valja pripaziti na nekoliko detalja, posebno ako niste iskusni korisnik naredbenog sučelja na vašoj platformi. Ako se kazalo s izvršnim okruženjem Pythona ne nalazi na popisu kazala s izvršnim programima (varijabla `PATH` naredbenog sučelja), morat ćemo navesti puni put do izvršnog programa `python3` (to će tipično biti slučaj na Windowsima). Nadalje, ako se pri zadavanju naredbe ne nalazimo u istom kazalu kao i datoteka s izvornim kôdom, u argumentu ćemo trebati zadati puni put

[9] Ako vam se ne sviđa IDLE, slobodno možete koristiti i neki drugi uređivač teksta. S obzirom na to da Python podrazumijeva kodni standard UTF8, dobri i besplatni izbor bili bi Notepad++ na Windowsima, gedit na Linuxu te XCode na Mac OS X-u.

[10] Naredbeno sučelje pod Windowsima (Command Prompt) koristi se 8-bitnim kodiranjem koje, za razliku od Unicodea, ne podržava sve međunarodne znakove. Pokušate li na standardni izlaz ispisati neki znak koji se ne može prikazati u dotičnom 8-bitnom kodiranju, Python će prijaviti pogrešku `UnicodeEncodeError`. Stoga, za ispis hrvatskih znakova treba kodnu stranicu podesiti na 852. Ovaj problem se ni na koji način neće odraziti na čitanje i ispisivanje međunarodnih znakova u datoteke.

do te datoteke. U nastavku navodimo potpuni primjer poziva pod Windowsima, pod pretpostavkom da je Python instaliran na podrazumijevanoj lokaciji `C:\Python34`, te da se datoteka s izvornim kôdom nalazi u matičnom kazalu korisnika `C:\Users\Pero`:

```
$ C:\Python34\Python C:\Users\Pero\drugi.py
```

Kao što smo naveli na početku ovog odjeljka, ovakvim pozivanjem možemo se koristiti i iz naredbenih skripti, te pri definiranju prečaca na radnoj površini grafičkog sučelja.

1.5. Dodatna programska podrška

Prethodni odjeljak sadrži preporučeno i najjednostavnije rješenje za isprobavanje primjera iz ove knjige. Međutim, ponekad ćemo u praksi htjeti instalirati i dodatnu programsku podršku koja će nam omogućiti da s Pythonom napravimo više, bolje i jače. Ovdje ćemo se ukratko upoznati s tim dodatnim programima, ali bez ambicije da ulazimo u previše detalja jer svaka od tih tema zaslužuje cijelo poglavlje (ako ne i zasebnu knjigu).

1.5.1. Alternativne inačice izvršnog okruženja Pythona

Službena inačica izvršnog okruženja Pythona je program u strojnom kôdu dobiven prevođenjem iz izvornog kôda pisanog u programskom jeziku C. Zbog toga je ponekad nazivamo CPython kako bismo naglasili razliku u odnosu na alternativne implementacije koje ćemo ukratko predstaviti u nastavku ovog odjeljka. CPython ima dvojaku funkciju: prevođenje izvornog kôda Pythona u međukôd te izvođenje programa interpretiranjem međukôda. CPython interpretira pythonski međukôd na način da dohvaća te potom izvršava jednu po jednu instrukciju međukôda, sve dok ne naiđe na instrukciju za povratak iz zadanog programa.

Izvorni kôd u Pythonu možemo izvoditi i bez CPythona, korištenjem alternativnih inačica izvršnog okruženja. Prva takva inačica koju ćemo spomenuti jest Jython, koji prevodi izvorni pythonski kôd u međukod za Javin virtualni stroj JVM (engl. *Java virtual machine*). Kao što bismo mogli i pretpostaviti, Jython je napisan u Javi. Glavna prednost u odnosu na CPython je mogućnost lakog povezivanja s ostalim jezicima pod JVM-om. Primjerice, programi u Jythonu mogu koristiti sve razrede koji su dostupni JVM-u, uključujući i elemente standardnog korisničkog sučelja Jave koje pružaju biblioteke Swing ili AWT. Glavni nedostatak Jythona je sporije praćenje razvoja novih mogućnosti Pythona: Jython trenutno (svibanj 2015.) podržava verziju 2.7 jezika, dok je Python 3 još uvijek u planovima za budućnost. Relativna performansa Jythona u odnosu na Python jako ovisi od programa do programa, ali se može reći da je u prosjeku otprilike istog reda veličine.

Sve što smo rekli za Jython i JVM, vrijedi i za IronPython, odnosno Microsoftov virtualni stroj CLR (engl. *common language runtime*). IronPython prevodi izvorni kôd Pythona u međukôd koji se može izvoditi na Microsoftovoj platformi .NET. Glavna prednost IronPythona je lako povezivanje s bibliotekama pisanim za jezik C♯, kao i mogućnost izvođenja u okviru preglednika koji podržavaju Silverlight. IronPython trenutno podržava verziju 2.6 jezika, a ima otprilike jednaku prosječnu performansu kao i Jython.

Najnovija alternativna implementacija izvršnog okruženja Pythona poznata je pod imenom PyPy (šećer smo ostavili za kraj). Glavna značajka PyPya je mogućnost dinamičkog prevođenja Pythona u optimirani strojni kôd računala domaćina. Time se izbjegava gubitak vremena uslijed interpretiranja, što dovodi do prosječnog ubrzanja od preko šest puta. PyPy trenutno podržava Python 3.2, što znači da je vrlo blizu cilja da postane općenito primjenljiva zamjena za CPython. Ipak, ovdje bismo htjeli naglasiti da je PyPy još uvijek u eksperimentalnoj fazi pa početnicima preporučamo da svoje programe ipak prvo ispitaju u klasičnom interpretiranom Pythonu.

1.5.2. Razvoj u okviru Eclipsea

Programe u Pythonu možemo razvijati i u okviru Eclipsea. Eclipse je jedno od najopširnijih i najpopularnijih integriranih okolina za razvoj programske podrške. Eclipse prvenstveno podržava Javu, ali uz pomoć dodatka PyDev njime se možemo koristiti i za razvoj programa u Pythonu. Eclipse omogućava praktično navigiranje po velikom stablu izvornog kôda, automatsko dopunjavanje djelomično napisanih imena, praćenje izvođenja programa (debagiranje), detekciju pogrešaka u trenutku pisanja programa i slične mogućnosti na koje su nas navikla moderna integrirana razvojna okruženja.

Instaliranje Eclipsea s PyDevom za početnike može biti zahtjevan zadatak. S obzirom na to da je Eclipse pisan u Javi, prvo trebamo instalirati izvršno okruženje Jave. Zatim trebamo instalirati Eclipse, te na samom kraju PyDev. U svakom od ovih koraka na različitim platformama može doći do različitih komplikacija. Odoljet ćemo iskušenju da te komplikacije popisujemo i preporučamo rješenja jer će do vremena kad ova knjiga bude tiskana ti problemi vjerojatno biti riješeni (a izronit će neki novi). Stoga ćemo motiviranom čitatelju preporučiti da uloži neko vrijeme na eksperimentiranje, te da se u slučaju problema obrati iskusnijim prijateljima i kolegama.

Iako Eclipse može značajno olakšati razvoj velikog programa, htjeli bismo naglasiti da ni u kojem slučaju ne može zamijeniti interaktivnu ljusku Pythona. Naprosto, pokazuje se da je interaktivno prototipiranje i ispitivanje dijelova programa najbolji način razvoja programa i učenja programskog jezika. Python izvanredno dobro podržava takav interaktivan pristup programiranju te ga je šteta ne iskoristiti!

1.5.3. Razvoj u okviru Visual Studija

Kao vrlo dobra alternativa Eclipseu, korisnicima na operacijskim sustavima Windows na raspolaganju je Microsoftov Visual Studio. Od verzije Visual Studio 2015 podrška za Python se nudi među opcijama već prilikom instalacije pa ne treba naknadno uključivati nikakve dodatke. Za učenje i nekomercijalni razvoj, dostupna je besplatna inačica pod nazivom Visual Studio Community Edition, koja se može preuzeti s Microsoftovih mrežnih stranica. Za korištenje Visual Studija potrebno je registrirati se, ali je ta registracija besplatna. Osim Visual Studija, neophodno je instalirati i odgovarajući Python interpretator i pripadajuće biblioteke.