Autor knjige Python bez oklevanja

Dok su napolju u šetnji, Paul zastaje i raspravlja o pravilnom izgovaranju reči „tuple" (n–torka) sa svojom mučenicom od žene.



Ovo je Dierdrena uobičajna reakcija. ☺

Paul Barry živi i radi u gradiću *Carlow* (u *Irskoj*), koji ima oko 35.000 stanovnika, a nalazi se oko 80km jugozapadno od glavnog grada, *Dublina*.

Paul ima diplomu *Informacionih sistema*, kao i magistaturu *Računarstva*. Takođe je završio postdiplomske studije *Predavanja i nastave*.

Paul radi u *Institutu za tehnologiju, Carlow* od 1995, i predaje na njemu od 1997. Pre nego što je počeo da predaje, Paul je proveo deset godina u IT industriji radeći u Irskoj i Kanadi, gde je najviše radio u okviru zdravstva. Paul je oženjen sa Deirdre i oni imaju troje dece (od kojih je dvoje sada na fakultetu).

Programski jezik Python i tehnologije vezane za njega bili su sastavni deo Paulovih predavanja za studente počevši od akademske godine 2007.

Paul je autor (ili koautor) još četiri tehničke knjige: dve o Pythonu i dve o *Perlu*. U prošlosti, mnogo je pisao za časopis *Linux Journal Magazine*, u kojem je bio dopisnik i urednik.

Paul je odrastao u *Belfastu*, u *Severnoj Irskoj*, što donekle objašnjava njegove stavove kao i njegov smešan akcenat (osim, naravno, ako ste i vi „sa Severa", pa su u tom slučaju Paulova shvatanja i akcenat *sasvim normalni*).

Pronaći ćete Paula na *Twitteru* (*@barrypj*), i na njegovom sajtu:
`http://paulbarry.itcarlow.ie`.

Sadržaj ukratko

1	Osnove: *Brzi početak*	1
2	Podaci u listi: *Rad sa uređenim podacima*	47
3	Strukturisani podaci: *Rad sa strukturisanim podacima*	95
4	Višekratna upotreba koda: *Funkcije i moduli*	145
5	Izgradnja veb aplikacije: *Sada stvarno*	195
6	Čuvanje i obrada podataka: *Gde da stavite svoje podatke*	243
7	Upotreba baze podataka: *Upotrebimo Pythonov DB-API*	281
8	Malo klase: *Apstrahovanje ponašanja i stanja*	309
9	Protokol za upravljanje kontekstom: *Uklapanje u Pythonovu naredbu with*	335
10	Dekoratori funkcija: *Umotavanje funkcija*	363
11	Obrada izuzetaka: *Šta da se radi kad stvari krenu naopako*	413
11¾	Nešto o nitima: *Bavimo se čekanjem*	461
12	Napredna iteracija: *Brze petlje*	477
A	Instalacija: *Instaliranje Pythona*	521
B	Pythonanywhere: *Isporuka vaše veb aplikacije*	529
C	Deset nepomenutih stvari: *Uvek ima još šta da se nauči*	539
D	Deset neobrađenih projekata: *Još neke alatke, biblioteke i moduli*	551
E	Uključite se: *Python zajednica*	563

Sadržaj (prava stvar)

Uvod

Vaš mozak i Python. Vi pokušavate nešto da naučite, dok vaš mozak hoće da vam pomogne tako što se trudi da ne zapamtite to što učite. Vaš mozak misli da je bolje ostaviti mesta za važnije stvari, na primer: „Koje divlje životinje treba izbegavati i da li je pametno ići na sankanje bez odeće." Dakle, kako ubediti svoj mozak da vam je od životne važnosti da naučite da programirate u jeziku Python?

Za koga je ova knjiga? xxiv

Već znamo šta mislite xxv

Mi znamo kako vaš *mozak* misli xxv

Metakognicija: razmišljanje o razmišljanju xxvii

Evo šta smo MI radili xxviii

Pročitajte xxx

Tim za tehnički pregled xxxii

osnove

1 Brzi početak

Počnite da programirate u jeziku Python što je pre moguće.
U ovom poglavlju, predstavljamo osnove programiranja u jeziku Python, a to radimo u tipičnom stilu serije *Bez oklevanja*: odmah uskačemo. Nakon prvih par stranica, izvršićete prvi primer programa. Na kraju poglavlja, nećete samo biti u stanju da izvršite primer programa, nego ćete i razumeti njegov kôd (i više od toga). Usput, učićete o nekoliko stvari po kojima se programski jezik **Python** odlikuje. Dakle, da više ne gubimo vreme. Okrenite stranicu, pa da počnemo!!

Understanding IDLE's Windows 4
Izvršavanje koda, naredbu po naredbu 8
Funkcije + Moduli = Standardna biblioteka 9
Strukture podataka su ugrađene 13
Pozivanjem metoda dobiju se rezultati. 14
Odlučivanje kada da se izvrše blokovi koda 15
Kakav „else" može da bude uz „if"?. 17
Svite mogu da sadrže ugnežđene svite 18
Povratak u Python Shell 22
Eksperimentisanje u prozoru interpretera 23
Ponavljanje nad sekvencom objekata 24
Iteriranje zadati broj puta 25
Primena rezultata zadatka 1 na naš kôd 26
Podešavanje pauziranja izvršenja 28
Generisanje slučajnih celih brojeva sa Pythonom 30
Kodiranje ozbiljne poslovne aplikacije. 38
Python kôd se lako čita. 39
Da li vas uvlačenje izluđuje? 40
Tražimo od Interpretera pomoć oko funkcije 41
Eksperimentisanje sa opsezima 42
Chapter 1's Code. 46

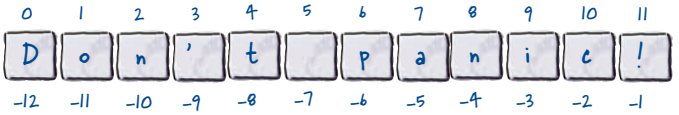


podaci u listi

2 Rad sa uređenim podacima

Svi programi obrađuju podatke, a Python programi nisu izuzetak. U stvari, osvrnite se: *podaci su svuda*. Veliki deo programiranja, ako ne i veći, bavi se podacima: pribavljanje podataka, obrada podataka, analiza podataka. Da biste efikasno radili sa podacima, morate negde da stavite svoje podatke dok ih obrađujete. Python u tom pogledu blista, zahvaljujući (velikim delom) tome što uključuje pregršt *široko primenjivih* struktura podataka: **liste**, **rečnike, n-torke** i **skupove**. U ovom poglavlju ćemo predstaviti sve četiri, pre nego što veći deo ovog poglavlja posvetimo udubljivanju u liste (a u sledećem poglavlju se udubljujemo u ostale tri strukture). Odmah obrađujemo strukture podataka, jer će se najveći deo onoga što ćete verovatno raditi sa Pythonom vrteti oko rada sa podacima.

Brojevi, stringovi... i objekti 48
Upoznajte četiri ugrađene strukture podataka 50
Neuređena struktura podataka: rečnik 52
Struktura podataka koja izbegava duplikate: skup 53
Definisanje liste 55
Upotrebite editor kada radite sa više od par redova koda 57
Lista „raste” za vreme izvršavanja 58
Provera članstva sa „in” 59
Uklanjanje objekata iz liste 62
Proširivanje liste objektima 64
Umetanje objekta u listu 65
Kako kopirati strukturu podataka 73
Liste proširuju notaciju sa uglastim zagradama 75
Liste shvataju Start, Stop, i Step 76
Start i Stop u listama 78
Isečci na poslu sa listama 80
Pythonove „for” petlje shvataju liste 86
Marvinovi isečci detaljno 88
Kada ne treba koristiti liste 91
Kôd poglavlja 2, 1 od 2 92



strukturisani podaci

3 Rad sa strukturisanim podacima

Pythonova lista je odlična struktura podataka, ali to nije rešenje za sve podatke. Kada imate stvarno strukturisane podatke (a lista nije najbolji izbor za njihovo čuvanje), Python vam priskače u pomoć sa svojim ugrađenim **rečnikom**. Odmah upotrebljiv, rečnik omogućava čuvanje svake kolekcije *parova ključ/vrednost* i njenu obradu. U ovom poglavlju ćemo dugo i temeljno razmatrati Pythonov rečnik, i – usput – pogledati i **skup** i **n-torku**. Zajedno sa **listom** (koju smo upoznali u prethodnom poglavlju), strukture podataka rečnik, skup i n-torka pružaju jedan skup alata za podatke koji pomažu da Python i podaci postanu jedna moćna kombinacija.

Rečnik čuva parove ključ/vrednost 96
Kako u kodu prepoznati rečnik 98
Redosled unošenja se NE zadržava 99
Pretraga vrednosti pomoću uglastih zagrada 100
Rad sa rečnicima za vreme izvršavanja 101
Ažuriranje mere frekvencije 105
Iteracija nad rečnikom 107
Iteracija nad ključevima i vrednostima. 108
Iteracija nad rečnikom putem „items” 110
Koliko su rečnici dinamični? 114
Izbegavanje greške KeyError za vreme izvršavanja 116
Provera članstva sa „in” 117
Obezbeđivanje inicijalizacije pre upotrebe 118
Zamena „not in” umesto „in” 119
Upotreba metoda „setdefault” 120
Efikasno pravljenje skupova 124
Korišćenje metoda skupova 125
Argumenti za n-torke 132
N-torke su nepromenljive 133
Pristupanje kompleksnoj strukturi podataka 141
Kôd poglavlja 3, 1 od 2 143



Višekratna upotreba koda

4 Funkcije i moduli

Višekratna upotreba koda je ključna za održiv sistem.
A kada je u Pythonu reč o višekratnoj upotrebi koda, sve počinje i završava se skromnom **funkcijom**. Uzmite par redova koda, dodelite im ime, i imate funkciju (koja može višekratno da se upotrebljava). Uzmite kolekciju funkcija i spakujte ih kao fajl, i imate **modul** (koji takođe može višekratno da se upotrebljava). Istina je kada kažu: dobro je deliti, a kada stignemo na kraj ovog poglavlja, vi ćete biti na dobrom putu da **delite** i **višekratno koristite** svoj kôd, zahvaljujući tome što ćete znati kako rade Pythonove funkcije i moduli.

Višekratno korišćenje koda sa funkcijama 146
Predstavljanje funkcija 147
Pozivanje funkcije 150
Funkcije mogu da prihvataju argumente 154
Vraćanje jedne vrednosti 158
Vraćanje više vrednosti 159
Podsećanje na ugrađene strukture podataka 161
Pravljenje opštije korisne funkcije 165
Pravimo još jednu funkciju, 1 od 3 166
Određivanje podrazumevanih vrednosti argumenata 170
Poziciono dodeljivanje i po ključnim rečima 171
Utvrdimo ono što već znamo o funkcijama 172
Izvršavanje Pythona sa komandne linije 175
Pravljenje obaveznih fajlova za postavku 179
Pravljenje fajla za distribuciju 180
Instaliranje paketa „pip” 182
Prikaz semantike poziva po vrednosti 185
Prikaz semantike poziva po referenci 186
Instaliranje alatki za testiranje 190
Koliko je naš kôd usklađen sa dokumentom PEP 8? 191
Razumevanje poruka o greškama 192
Kôd poglavlja 4 194



module

izgradnja veb aplikacije

5 Sada stvarno

U ovoj fazi, već dovoljno znate Python da postajete opasni.
Pošto ste prešli prva četiri poglavlja ove knjige, sada ste u stanju da produktivno koristite Python u okviru bilo koje oblasti primene (iako ima još mnogo da se uči o Pythonu). Umesto da razmatramo dugačak spisak tih oblasti primene, mi ćemo u ovom i u sledećim poglavljima učenje oslanjati na razvoju jedne aplikacije smeštene na Veb, pošto je to područje na kojem je Python posebno jak. Usput ćete učiti još malo o Pythonu. Međutim, pre nego što krenemo dalje, hajde da ukratko ponovimo ono što već znate o Pythonu..

Python: šta već znate 196
Šta želimo da naša veb aplikacija radi? 200
Hajde da instaliramo Flask. 202
Kako radi Flask?? 203
Prvo izvršavanje vaše Flask veb aplikacije 204
Pravljenje objekta Flask veb aplikacije 206
Dekorisanje funkcije URL-om 207
Izvršavanje ponašanja vaše veb aplikacije 208
Izlaganje funkctionalnosti na Vebu 209
Izrada HTML obrasca 213
Šabloni vezani za veb stranice 216
Vizualizacija šablona kroz Flask 217
Prikaz HTML obrasca naše veb aplikacije. 218
Priprema da se izvrši kôd šablona 219
Shvatanje HTTP statusnih kodova 222
Obrada poslatih podataka 223
Usavršavanje ciklusa uredi/stani/pokreni/testiraj. 224
Pristup podacima HTML obrasca pomoću Flaska 226
Upotreba podataka iz zahteva u vašoj veb aplikaciji. 227
Prikazivanje rezultata kao HTML 229
Priprema vaše veb aplikacije za oblak 238
Kôd poglavlja 5 241



čuvanje i obrada podataka

6 Gde da stavite svoje podatke

Kad tad, moraćete negde bezbedno da čuvate svoje podatke.
A kad je reč o **čuvanju podataka**, Python ima rešenje i za to. U ovom poglavlju, učićete o čuvanju podataka u *tekstualnim fajlovima* i uzimanju podataka iz njih, koji – kao mehanizam za skladištenje – možda deluju pomalo prosto, ali se i pored toga koriste u mnogim problemskim područjima. Pored spremanja i uzimanja podataka iz fajlova, učićete takođe i neke tajne zanata kada je u pitanju obrada podataka. „Ozbiljne stvari" (čuvanje podataka u bazi podataka) ostavljamo za sledeće poglavlje, ali i za sada imamo dovoljno da nas zaokupi dok radimo sa fajlovima.

Uradimo nešto sa podacima vaše veb aplikacije 244
Python podržava otvaranje, obradu, zatvaranje 245
Čitanje podataka iz postojećeg fajla 246
„with" je bolje od otvaranje, obrada, zatvaranje. 248
Pregledanje dnevnika kroz vašu veb aplikaciju 254
Ispitajte sirove podatke sa View Source 256
Vreme za izbegavanje 257
Pregledanje celog dnevnika u vašoj veb aplikaciji 258
Evidentiranje konkretnih atributa veb zahteva 261
Evidentiranje jednog reda razgraničenih podataka 262
Od sirovih podataka do čitljivog izlaza 265
Pravljenje čitljivog izlaza sa HTML-om 274
Ugradite logiku prikaza u svoj šablon 275
Pravljenje čitljivog izlaza sa Jinja2 276
Trenutno stanje koda naše veb aplikacije 278
Postavljanje pitanja o podacima 279
Kôd poglavlja 6 280

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	{'e', 'i'}

upotreba baze podataka

7 Upotrebimo Pythonov DB-API

Čuvanje podataka u sistemu relacione baze podataka je praktično. U ovom poglavlju, učićete kako da pišete kôd koji komunicira sa popularnom tehnologijom baza podataka **MySQL**, pomoću jednog generičkog API-ja za baze podataka po imenu **DB-API**. DB-API (koji se standardno isporučuje u svakoj Python instalaciji) vam omogućava da pišete kôd koji se lako prenosi iz jednog u drugi proizvod baze podataka...pod uslovom da te baze podataka razumeju SQL. Mada ćemo mi koristiti MySQL, ništa vas ne sprečava da koristite DB-API kôd sa svojom omiljenom relacionom bazom podataka. Hajde da vidimo šta je sve potrebno da bi se relaciona baza podataka koristila iz Pythona. U ovom poglavlju nećemo naučiti mnogo novog Pythona, ali koristiti Python za rad sa bazama podataka je **vrlo korisno**, pa ga vredi naučiti.

Osposobiti veb aplikaciju za baze podataka 282

Zadatak 1: Instalacija MySQL servera. 283

Pythonov DB-API 284

Zadatak 2: Instaliranje MySQL drajvera baze podataka za Python 285

Instaliranje MySQL-Connector/Python-a 286

Zadatak 3: Pravljenje baze podataka i tabela naše veb aplikacije . 287

Biranje strukture za evidentirane podatke 288

Provera da li je tabela spremna za podatke 289

Zadatak 4: Pisanje koda za rad sa bazom podataka i tabelama naše veb aplikacije 296

Smeštanje podataka je samo pola posla 300

Kako najbolje višekratno koristiti kôd vaše baze podataka? . . .301

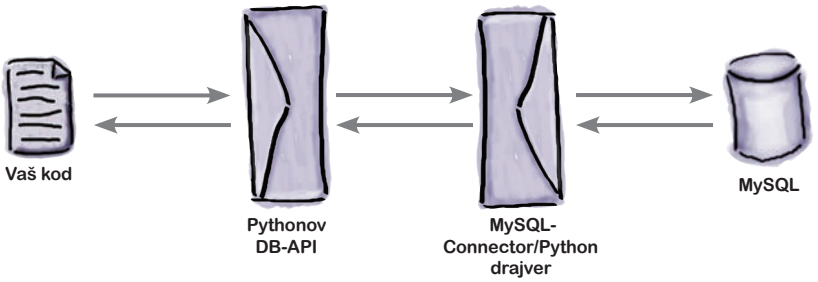
Razmotrite šta pokušavate višekratno da koristite 302

A šta je sa uvoženjem? 303

Videli ste taj šablon i pre 305

To što je loše ipak nije tako strašno 306

Kôd poglavlja 7 307

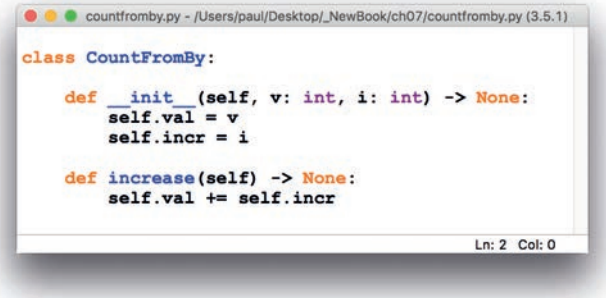


malo klase

8 Apstrahovanje ponašanja i stanja

Klase omogućavaju da se ponašanje koda spoji sa stanjem.
U ovom poglavlju, ostavljate svoju veb aplikaciju po strani dok učite o pravljenju Python **klasa**. To radite da biste stekli znanje potrebno da se menadžer konteksta napravi pomoću Pythonove klase. Pošto je pravljenje i upotreba klasa inače toliko korisna stvar, posvećujemo im ovo poglavlje. Nećemo reći sve o klasama, ali ćemo se dotaći svih elemenata koje morate da shvatite da biste sa sigurnošću napravili menadžer konteksta na koji vaša veb aplikacija čeka. Krenimo, pa da vidimo šta sve tu ima.

Uklapanje sa naredbom „with” 310
Osnove objektnog orijentisanja 311
Pravljenje objekata od klasa 312
Objekti dele ponašanje ali ne i stanje 313
Radimo više sa CountFromBy 314
Pozivanje metoda: ulaženje u detalje. 316
Dodavanje metoda u klasu 318
Koliko je važan „self” 320
Shvatanje opsega 321
Prefiks imena atributa je uvek „self” 322
Inicijalizujte vrednosti (atributa) pre upotrebe 323
„init” sa crticama inicijalizuje atribute 324
Inicijalizovanje atributa – „init” sa crticama 325
Prikazivanje klase CountFromBy 328
Definisanje predstavljanja klase CountFromBy 329
Razumne podrazumevane vrednosti za CountFromBy 330
Klase: šta već znamo 332
Kôd poglavlja 8 333



protokol za upravljanje kontekstom

9 Uklapanje u Pythonovu naredbu with

Vreme je da ovo što ste upravo naučili i upotrebite.
U poglavlju 7 bilo je reči o upotrebi **relacione baze podataka** sa Pythonom, dok smo u poglavlju 8 imali uvod u korišćenje **klasa** u vašem Python kodu. U ovom poglavlju se obe te tehnike kombinuju da bi se napravio **menadžer konteksta** koji će omogućiti da naredbu `with` proširimo na rad sa sistemima relacionih baza podataka. U ovom poglavlju, uklopićete se u naredbu `with` pomoću nove klase, usklađene sa Pythonovim **protokolom za upravljanje kontekstom**.

Koji je najbolji način da se deli kôd baze podataka naše veb aplikacije? 336

Razmotrite šta pokušavate da uradite, ponovo 337

Upravljanje kontekstim pomoću metoda 338

Već ste videli jedan menadžer konteksta na delu 339

Pravljenje nove klase menadžera konteksta 340

Inicijalizujte klasu sa dbconfig 341

Podešavanje u „enter” sa crticama 343

Rušenje metodom „exit” sa crticama 345

Ponovno razmatranje koda vaše veb aplikacije, 1 od 2 348

Prisećanje na funkciju „log_request”. 350

Ispravljanje funkcije „log_request”. 351

Podsećanje na funkciju „view_the_log” 352

Ne menja se samo kôd 353

Ispravljanje funkcije „view_the_log”. 354

Odgovori na pitanja o podacima. 359

Kôd poglavlja 9, 1 od 2 360


```
File  Edit  Window  Help  Checking our log DB
$ mysql -u vsearch -p vsearchlogDB
Enter password:

Welcome to MySQL monitor...


mysql> select * from log;
+----+---------------------+-----------------------+---------+-----------+----------------+-----------------+
| id | ts                  | phrase                | letters | ip        | browser_string | results         |
+----+---------------------+-----------------------+---------+-----------+----------------+-----------------+
|  1 | 2016-03-09 13:40:46 | life, the uni ... ything | aeiou   | 127.0.0.1 | firefox        | {'u', 'e', 'i', 'a'} |
|  2 | 2016-03-09 13:42:07 | hitch-hiker           | aeiou   | 127.0.0.1 | safari         | {'i', 'e'}      |
|  3 | 2016-03-09 13:42:15 | galaxy                | xyz     | 127.0.0.1 | chrome         | {'y', 'x'}      |
|  4 | 2016-03-09 13:43:07 | hitch-hiker           | xyz     | 127.0.0.1 | firefox        | set()           |
+----+---------------------+-----------------------+---------+-----------+----------------+-----------------+
4 rows in set (0.0 sec)


mysql> quit
Bye
```

dekoratori funkcija

10 Umotavanje funkcija

Kada treba unaprediti kôd, protokol za upravljanje kontekstom iz poglavlja 9 nije jedini izbor. Python vam takođe omogućava da koristite **dekoratore** funkcija, tehniku putem koje možete da dodajete kôd u postojeću funkciju *bez* potrebe da se išta menja u postojećem kodu funkcije. Ako vam se čini da to zvuči kao nekakva crna magija, ne očajavajte: nije u tome stvar. Međutim, pravljenje dekoratora funkcije mnogi Python programeri često smatraju teškom tehnikom kodiranja, pa se zato ne koristi onoliko često koliko bi trebalo. U ovom poglavlju, planiramo da vam pokažemo da, uprkos tome što je reč o naprednoj tehnici, nije toliko teško napraviti i koristiti vlastite dekoratore.

Vaš veb server (a ne vaš računar) izvršava vaš kôd 366
Flaskova tehnologija sesije dodaje stanje 368
Pretraživanje rečnika vraća vrednost. 369
Upravljanje prijavljivanjem pomoću sesija. 374
Hajde da pravimo Logout i proveru Statusa 377
Kako proslediti funkciju drugoj funkciji 386
Pozivanje prosleđene funkcije 387
Prihvatanje liste argumenata 390
Obrada liste argumenata. 391
Prihvatanje rečnika argumenata 392
Obrada rečnika argumenata 393
Prihvatanje svakog broja i tipa argumenata funkcije. 394
Pravljenje dekoratora funkcije 397
Završna tačka: rukovanje argumentima 401
Stavljanje vašeg dekoratora u rad 404
Povratak na ograničavanje pristupa za /viewlog. 408
Kôd poglavlja 10, 1 od 2 410



obrada izuzetaka

11 Šta da se radi kad stvari krenu naopako

Stvari kreću naopako, stalno – bez obzira na to koliko je vaš kôd dobar. Uspešno ste uradili sve primere u ovoj knjizi, pa ste verovatno uvereni da sav do sada prikazani kôd funkcioniše. Ali da li to znači da je taj kôd robustan? Verovatno nije. Pisati kôd polazeći od pretpostavke da se nikada neće desiti ništa loše je (u najboljem slučaju) naivno. U najgorem slučaju, to je opasno, pošto nepredviđene stvari mogu da se dogode (i događaju se). Mnogo je bolje da kada kodirate budete oprezni, a ne previše samouvereni. Potrebna je obazrivost da bi vaš kôd radio ono što želite da on uradi, kao i da pravilno reaguje ako stvari krenu naopako. U ovom poglavlju, nećete videti samo šta sve može da krene naopako, nego ćete takođe naučiti šta da uradite kada se to desi (a često i pre nego što se desi).

Baze podataka nisu uvek dostupne 418

Napadi sa Veba su prava muka 419

Ulaz/izlaz je (ponekad) spor 420

Vaš poziv funkcije može da ne uspe 421

Za kôd sklon greškama uvek koristite Try 423

try jednom, ali except više puta 426

Hendler koji hvata sve izuzetke 428

Učenje o izuzecima iz „sys” 430

Hendler koji hvata sve izuzetke, ponovo 431

Vraćanje na kôd naše veb aplikacije 433

Tiha obrada izuzetaka 434

Obrada drugih grešaka baze podataka 440

Izbegavajte tesno spojen kôd 442

Modul DBcm, ponovo 443

Pravljenje namenskih izuzetaka 444

Šta još može da bude problem sa kodom „DBcm”? 448

Obrada za SQLError je drugačija 451

Podizanje izuzetka SQLError 453

Brza rekapitulacija: dodavanje robustnosti 455

Kako izaći na kraj sa čekanjem? Zavisi... 456

Kôd poglavlja 11, 1 od 3 457


```
...
Exception
     +-- StopIteration
     +-- StopAsyncIteration
     +-- ArithmeticError
     |    +-- FloatingPointError
     |    +-- OverflowError
     |    +-- ZeroDivisionError
     +-- AssertionError
     +-- AttributeError
     +-- BufferError
     +-- EOFError
          ...
```

nešto o nitima

11 3/4 Bavimo se čekanjem

Vaš kôd se ponekad dugo izvršava.
Zavisno od toga ko primećuje, to može i ne mora da bude problem. Ako nekom kodu treba 30 sekundi da obavi svoj posao „iza scene", to čekanje ne mora da bude problem. Međutim, ako vaš korisnik čeka da vaša aplikacija odgovori, pa to potraje 30 sekundi, svako će da primeti. Šta treba da uradite da biste rešili ovaj problem, zavisi od toga šta pokušavate da uradite (i ko je taj koji čeka). U ovom kratkom poglavlju, ukratko opisujemo neke opcije, zatim razmatramo jedno rešenje na aktuelno pitanje: *šta se dešava ako nešto previše traje?*

Čekanje: šta da se radi? 462
Kakvi su vam upiti nad bazom podataka? 463
INSERT i SELECT se razlikuju 464
Raditi više stvari u isto vreme 465
Nemojte da se prestravite: koristite niti 466
Idemo redom: bez panike 470
Ne brinite o maleru: Flask može da pomogne 471
Da li je vaša veb aplikacija sada robustna? 474
Kôd poglavlja 11¾, 1 od 2. 475
Kôd poglavlja 11¾, 2 od 2. 476



napredna iteracija

12 Brze petlje

Često je neverovatno koliko vremena naši programi provedu u petljama. To ne iznenađuje, pošto većina programa postoji zato da bi veoma brzo nešto radila mnogo puta. Kada je u pitanju optimizovanje petlji, postoje dva pristupa: (1) unaprediti sintaksu petlji (da bi se petlje lakše definisale), i (2) unaprediti način izvršavanja petlji (da bi one bile brže). U ranim fazama Pythona 2 (to jest, *jako*, *jako* davno), dizajneri jezika su dodali jedan element jezika koji implementira oba pristupa, a on ima jedno pomalo čudno ime: **comprehension – skraćen način generisanja programskim putem**. Ali nemojte dozvoliti da vas to čudno ime odbije: kad budete proradili ovo poglavlje, vi ćete se čuditi kako ste do sada uspevali da živite bez toga.



Čitati CSV podatke kao liste 479
Čitati CSV podatke kao rečnike 480
Strip, zatim Split na sirovim podacima 482
Pazite kada ulančavate pozive metoda 483
Transformisanje podataka u format koji vam je potreban 484
Transformisanje u rečnik listi 485
Uočavanje šablona za liste 490
Konvertovanje šablona u skraćeni oblik 491
Pogledajte skraćeno generisanje malo bolje 492
Skraćeno generisanje rečnika 494
Proširite skraćeno generisanje filterima 495
Rešavanje komplikacije na Pythonov način 499
Skraćeno generisanje skupova na delu 505
Šta je sa „skraćenim generisanjem n-torki”? 507
Zagrada oko koda == generator 508
Skraćeno generisanje liste u obradi URL-a 509
Obrada URL-a pomoću generatora 510
Definišite šta vaša funkcija treba da radi 512
Povinujte se sili generatorskih funkcija 513
Praćenje generatorske funkcije, 1 od 2 514
Jedno završno pitanje 518
Kôd poglavlja 12 519
Vreme je da se ide… 520

instalacija

a

Instaliranje Pythona

Idemo redom: hajde da instaliramo Python na vaš računar. Bilo da koristite *Windows*, *Mac OS X*, ili *Linux*, Python je sve predvideo. Način instaliranja na svakoj od tih platformi je specifičan za način rada svakog od tih operativnih sistema (znamo...užas, zar ne?), a Pythonova zajednica se trudi da obezbedi instalere za sve popularne sisteme. U ovom kratkom dodatku, vodimo vas kroz instaliranje Pythona na vaš računar.

Instaliramo Python 3 na Windows 522
Isprobajte Python 3 na Windowsu 523
Dodaci za Python 3 na Windowsu 524
Instalirajte Python 3 na Mac OS X (macOS) 525
Provera i konfigurisanje Pythona 3 za Mac OS X 526
Instalirajte Python 3 na Linux 527

b

PythonAnywhere

Isporuka vaše veb aplikacije

Na kraju poglavlja 5, izneli smo tvrdnju da za isporuku vaše veb aplikacije na oblak treba svega 10 minuta. Sada je vreme da ispunimo to obećanje. U ovom dodatku, provešćemo vas kroz proces isporučivanja vaše veb aplikacije na *PythonAnywhere*, od početka do kraja za oko 10 minuta. *PythonAnywhere* je omiljen u zajednici Python programera, a nije ni teško videti zašto:radi tačno onako kao što očekujete, ima odličnu podršku za Python (i Flask), a – što je najbolje – možete da počnete sa iznajmljivanjem svoje veb aplikacije sasvim besplatno. Hajde da pregledamo *PythonAnywhere*..

Tačka 0: malo pripreme 530
Tačka 1: prijavite se na PythonAnywhere 531
Tačka 2: otpremite vaše fajlove na oblak. 532
Tačka 3: izdvojite i instalirajte svoj kôd 533
Tačka 4: napravite početnu veb aplikaciju, 1 od 2. 534
Tačka 5: konfigurišite svoju veb aplikaciju 536
Tačka 6: isprobajte svoju veb aplikaciju koja se nalazi u oblaku! . 537

deset nepomenutih stvari



Uvek ima još šta da se nauči

Nismo ni imali nameru da pokušamo da obuhvatimo sve.
Cilj ove knjige je bio da vam pokažemo dovoljno Pythona da bismo vas što je brže moguće stavili u pogon. Mogli smo da obuhvatimo još mnogo više, ali nismo. U ovom dodatku, opisujemo najvažnijih 10 stvari koje smo – da smo imali još nekih 600 stranica – mogli da obradimo. Neće vas svih tih 10 stvari zanimati, ali pregledajte ih na brzinu da biste proverili da li smo slučajno pogodili nešto što ste želeli, ili pružili odgovor na neko pitanje koje vas muči. Sve programske tehnologije u ovom dodatku su ugrađene u Python i njegov interpreter.

1. A Python 2? 540
2. Virtuelna programska okruženja 541
3. Više o objektnom orijentisanju 542
4. Formati za stringove i slično 543
5. Sortiranje stvari 544
6. Više iz standardne biblioteke 545
7. Istovremeno izvršavanje koda 546
8. Grafički korisnički interfejsi – Tkinter (i zabava sa Turtles). . 547
9. Nije gotovo dok nije testirano 548
10. Otklanjanje grešaka, opet i opet 549

deset neobrađenih projekata

Još neke alatke, biblioteke i moduli

Znamo šta ste pomislili kad ste videli naslov ovog dodatka.
Zbog čega nisu prošli dodatak nazvali: *Dvadeset nepomenutih stvari*? Zašto sad *još* 10? U prošlom dodatku, ograničili smo diskusiju na ono što je već ugrađeno u Python (ono zbog čega se kaže da se jezik „isporučuje sa baterijama"). U ovom dodatku, bacamo udicu mnogo dalje, opisujemo gomilu tehnologija koje su vam na raspolaganju *zbog toga što* Python postoji. Tu ima mnogo dobrih stvari, pa – isto kao za prošlo poglavlje – brzi pregled vam neće *ni najmanje* škoditi.

1. Alternative za prompt >>> 552
2. Alternative za IDLE 553
3. Jupyter Notebook: IDE okruženje na Vebu 554
4. Nauka o podacima 555
5. Tehnologije za veb razvoj 556
6. Rad sa veb podacima 557
7. Još neki izvori podataka 558
8. Alatke za programiranje 559
9. Kivy: naš predlog za „najbolji projekat svih vremena" 560
10. Alternativne implementacije 561

uključite se



Python zajednica

Python je mnogo više od odličnog programskog jezika.
To je takođe jedna odlična zajednica. Python zajednica je gostoljubiva, raznolika, otvorena, druželjubiva, ona deli i daje. Prosto nas čudi da do danas nikome nije palo na pamet da to stavi na čestitku! Ozbiljno, programiranje u Pythonu nije samo jezik. Oko Pythona je nastao ceo ekosistem, u obliku izvrsnih knjiga, blogova, veb sajtova, konferencija, sretanja, korisničkih grupa i ličnosti. U ovom dodatku, mi snimamo Python zajednicu da vidimo šta ona nudi. Nemojte samo da sedite i programirate sami: **uključite se**!

BDFL: Benevolent Dictator for Life 564
Tolerantna zajednica: poštovanje raznovrsnosti 565
Python potkasti. 566
Zen of Python 567

1 osnove

Brzi početak



Počnite da programirate u jeziku Python što je pre moguće.

U ovom poglavlju, predstavljamo osnove programiranja u jeziku Python, a to radimo u tipičnom stilu serije *Bez oklevanja*: odmah uskačemo. Nakon prvih par stranica, izvršićete prvi primer programa. Na kraju poglavlja, nećete samo biti u stanju da izvršite primer programa, nego ćete i razumeti njegov kôd (i više od toga). Usput, učićete o nekoliko stvari po kojima se programski jezik **Python** odlikuje. Dakle, da više ne gubimo vreme. Okrenite stranicu, pa da počnemo!!

Raskid sa tradicijom

Kad uzmete bilo koju knjigu o nekom programskom jeziku, prvo što ćete videti je primer *Hello World*.





Ne, nećemo.

Ovo je knjiga iz serije *Bez oklevanja*, a mi ovde radimo drugačije. U drugim knjigama, običaj je da se počne od opisa programa *Hello World* u dotičnom jeziku. Međutim, sa Pythonom biste imali samo jednu naredbu kojom se poziva Pythonova ugrađena funkcija `print`, koja bi prikazala uobičajenu poruku „Hello, World!" na ekranu. Skoro da je previše uzbudljivo... a ne biste naučili takoreći ništa.

Dakle, ne, nećemo da vam pokažemo program *Hello World* u jeziku Python, pošto zaista ništa ne biste iz njega naučili. Krenućemo drugim putem...

Krenućemo od žešćeg primera

Naš plan za ovo poglavlje je da počnemo od jednog primera koji je donekle veći, pa je prema tome korisniji nego primer *Hello World*.

Odmah ćemo vam priznati da je naš primer pomalo *neprirodan*: on radi nešto, ali na duge staze možda i nije naročito koristan. Priznajemo to, ali odabrali smo takav primer koji će nam omogućiti da u što je moguće kraćem vremenu obradimo što više elemenata jezika Python. A obećavamo da ćete, kada budete proradili taj prvi primer programa, znati dovoljno da biste mogli da napišete *Hello World* u jeziku Python bez naše pomoći.

Odmah uskačemo

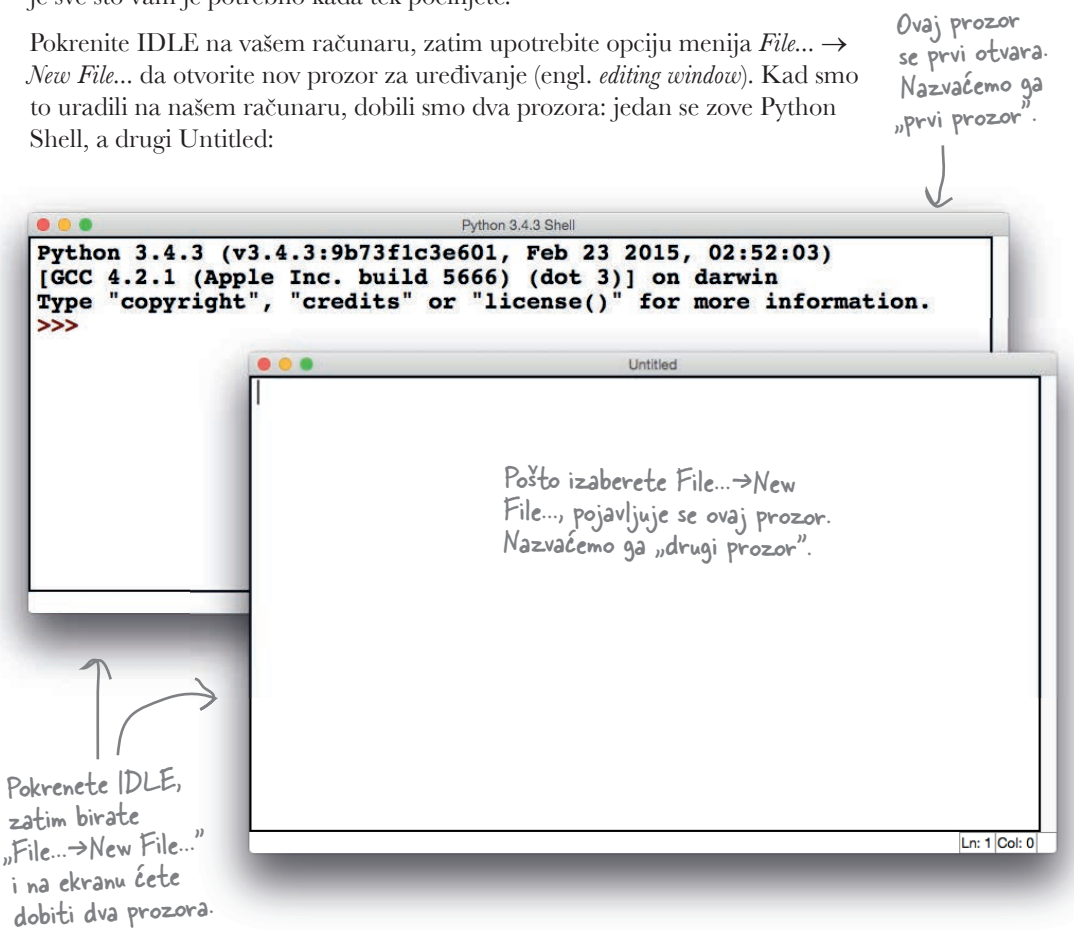
Ako još niste instalirali neku verziju Pythona 3 na svoj računar, sada zastanitei pređite na Dodatak A gde se nalaze uputstva za instaliranje korak po korak (utrošićete samo par minuta, obećavam).

Pošto instalirate najnoviji Python 3, imate sve što je potrebno da biste počeli da programirate Python, a kao pomoć u tome – za sada – koristićemo Pythonovo ugrađeno IDE (integrated development environment) okruženje.

Pythonov IDLE je sve što vam je potrebno

Kada na svoj računar instalirate Python 3, dobijate i jedno jednostavno ali upotrebljivo IDE okruženje po imenu IDLE. Mada ima mnogo različitih načina da se izvršava Python kôd (a upoznaćete ih mnogo u ovoj knjizi), IDLE je sve što vam je potrebno kada tek počinjete.

Pokrenite IDLE na vašem računaru, zatim upotrebite opciju menija *File...* → *New File...* da otvorite nov prozor za uređivanje (engl. *editing window*). Kad smo to uradili na našem računaru, dobili smo dva prozora: jedan se zove Python Shell, a drugi Untitled:



Understanding IDLE's Windows

Oba ova IDLE prozora su važna.

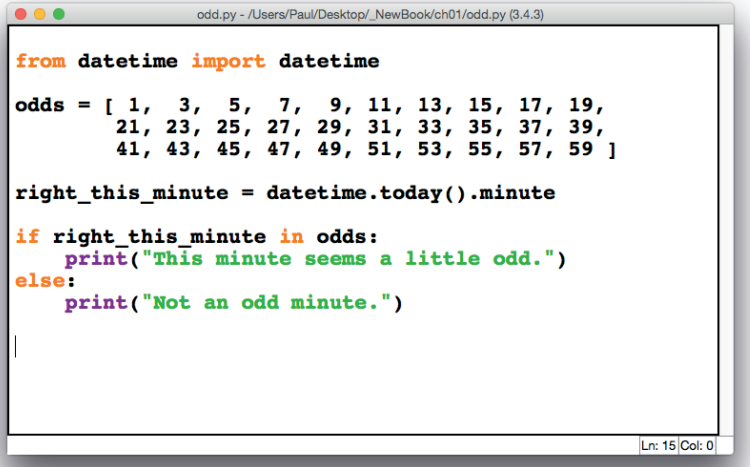
Prvi prozor, Python Shell, je jedno REPL okruženje koje se koristi za izvršavanje isečaka Python koda, obično ćete izvršavati jednu po jednu naredbu. Što duže budete koristili Python, sve ćete više voleti Python Shell, i mnogo ćete ga koristiti dok budete napredovali kroz ovu knjigu. Međutim, za sada nas više zanima drugi prozor.

Drugi prozor, Untitled, je prozor za uređivanje teksta koji može da se koristi za pisanje celih Python programa. To nije najbolji editor na svetu (pošto ta čast pripada *<upišite ovde ime vašeg omiljenog tekst editora>*), ali IDLE editor je prilično upotrebljiv i ima gomilu ugrađenih modernih mogućnosti, uključujući isticanje delova sintakse bojom (engl. *color-syntax handling)* i slično.

Pošto mi odmah uskačemo, hajde da upišemo jedan mali Python program u ovaj prozor. Kada završite sa upisivanjem sledećeg koda, upotrebite opciju menija *File...* → *Save...* da sačuvate svoj program pod imenom `odd.py`.

Pazite da upišete kôd *tačno* kao što stoji ovde:

Ne brinite za sada šta ovaj kôd radi. Samo ga prepišite u prozor za uređivanje. Ne zaboravite da ga sačuvate kao „odd.py" pre nego što krenete dalje.



Pa... *šta sad?* Ako ste bar malo kao mi, jedva čekate da izvršite ovaj kôd, zar ne? Hajde da to sada uradimo. Sa kodom u prozoru za uređivanje (kao što je prikazano gore), pritisnite taster F5 na tastaturi. Mogu se dogoditi različite stvari...



Za štrebere

Šta znači REPL?

To je štreberska skraćenica za „read-eval-print-loop", (petlja čitaj-izračunaj-štampaj) i opisuje jednu interaktivnu programersku alatku koja omogućava da do mile volje eksperimentišete sa isečcima koda. Saznaćete mnogo više nego što vam je potrebno ako posetite *http://en.wikipedia.org/wiki/Read-eval-print_loop*.

Šta se dalje događa...

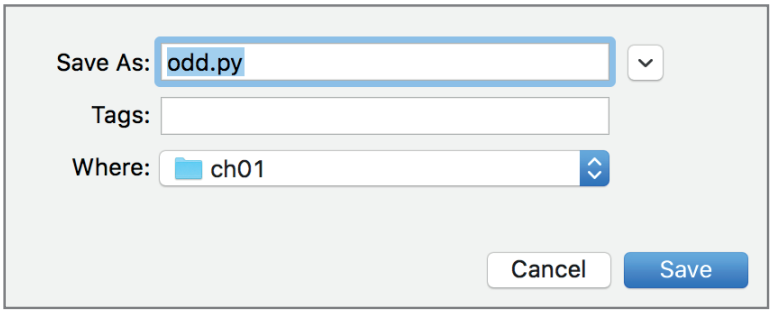
Ako se vaš kôd izvršio bez greške, pređite na sledeću stranu, i *nastavite*.

Ako ste zaboravili da sačuvate svoj kôd *pre* nego što ste pokušali da ga izvršite, IDLE se buni, pošto svaki novi kôd morate *najpre* da sačuvate u fajl. Ako niste sačuvali kôd, ugledaćete poruku sličnu ovoj:



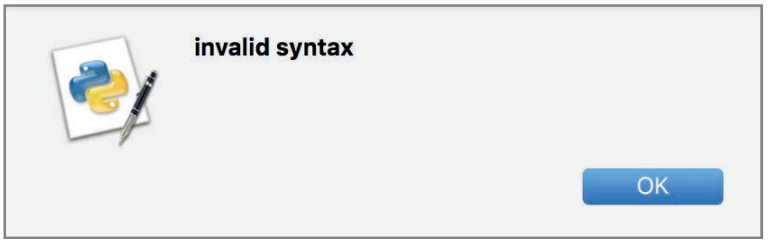
Podrazumevano, IDLE neće da izvrši kôd koji nije sačuvan.

Pritisnite dugme OK, zatim odredite ime za vaš fajl. Mi smo za ime fajla izabrali `odd`, i dodali smo ekstenziju `.py` (pošto je to Pythonova konvencija koje se vredi držati):



Vi možete slobodno za svoj program da izaberete bilo koje ime, ali verovatno je najbolje – ako pratite tekst – da koristite isto ime kao mi.

Ako se vaš kôd sada izvršio (pošto ste ga sačuvali), pređite na sledeću stranu, i *nastavite*. Međutim, ako negde u kodu imate grešku sintakse, ugledaćete ovu poruku:



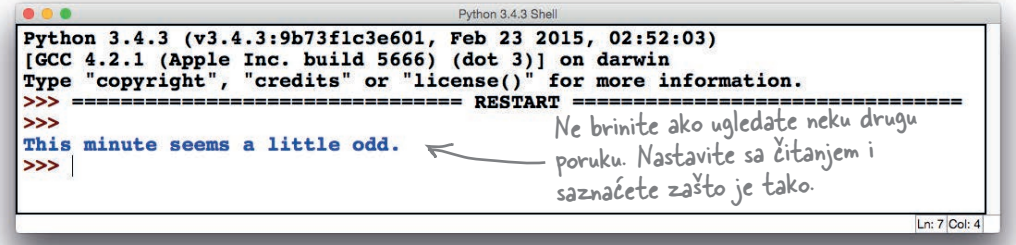
Kao što bez sumnje vidite, IDLE nije naročito vešt da kaže koja je to greška sintakse. Ali pritisnite OK, pa će jedan veliki crveni okvir da ukaže na mesto na kojem IDLE misli da se problem nalazi.

Pritisnite dugme OK, zatim pogledajte gde IDLE misli da se greška sintakse nalazi: tražite veliki crveni okvir u prozoru za uređivanje. Proverite da li je vaš kôd tačno isti kao naš, sačuvajte ponovo svoj fajl, a zatim pritisnite F5 i tako zatražite da IDLE ponovo izvrši vaš kôd.

Pritisnite F5 za izvršavanje koda

Pritisak na F5 izvršava kôd u trenutno izabranom IDLE prozoru za uređivanje teksta – naravno, pod pretpostavkom da vaš kôd ne sadrži grešku pri izvršavanju. Ako imate grešku pri izvršavanju, videćete **Traceback** poruku o grešci (crvene boje). Pročitajte poruku, zatim se vratite u prozor za uređivanje i proverite da li je kôd koji ste uneli tačno isti kao naš. Sačuvajte ispravljeni kôd, zatim ponovo pritisnite F5. Kada smo pritisnuli F5, Python Shell je postao aktivni prozor, i evo šta smo videli:

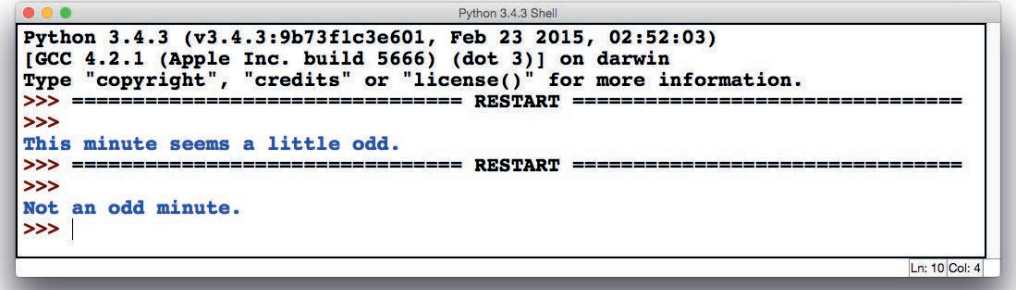
Od sada pa nadalje, mi ćemo „IDLE prozor za uređivanje teksta" zvati jednostavno „prozor za uređivanje".



Ne brinite ako ugledate neku drugu poruku. Nastavite sa čitanjem i saznaćete zašto je tako.

Zavisno od toga koliko je sati, mogli ste umesto ove poruke da ugledate poruku *Not an odd minute*. Ne brinite ako jeste, pošto ovaj program prikazuje jednu od te dve poruke zavisno od toga da li je vrednost minuta u tekućem vremenu vašeg računara neparan broj (engl. *odd*) (pa, rekli smo da je ovaj primer *neprirodan*, zar ne?). Ako sačekate jedan minut, zatim pritisnete prozor za uređivanje da biste ga izabrali, pa ponovo pritisnete F5, vaš kôd će se ponovo izvršiti. Ovog puta ćete videti drugu poruku (pod uslovom da ste sačekali zahtevani minut). Slobodno izvršavajte ovaj kôd koliko god često želite. Evo šta smo videli kada smo mi (vrlo strpljivo) sačekali zahtevani minut:

Pritisak na F5 dok ste u prozoru za uređivanje izvršava vaš kôd, zatim prikazuje rezutat u Python Shellu.



Hajde da neko vreme posvetimo učenju kako se ovaj kôd izvršava.

Kôd se izvršava odmah

Kada IDLE zatraži da Python izvrši kôd koji se nalazi u prozoru za uređivanje, Python počne na vrhu fajla i smesta počne sa izvršavanjem.

Za one od vas koji prelaze na Python sa nekog od jezika koji su kao C, obratite pažnju na to da u jeziku Python ne postoji pojam funkcije ili metoda `main()`. Takođe ne postoji ni pojam uobičajenog procesa uredi-prevedi-poveži-izvrši (engl. *edit-compile-link-run*). Kad koristite Python, vi svoj kôd uredite, sačuvate i *odmah* izvršite.



Stanite malo. Rekli ste „IDLE zahteva da Python izvrši kôd"...ali zar nije Python programski jezik, a IDLE njegovo IDE okruženje? Ako je tako, ko u stvari ovde sprovodi izvršavanje?!?

Oh, u pravu ste. To zbunjuje.

Evo šta morate da znate: „Python" je ime programskog jezika, a „IDLE" je ime Pythonovog ugrađenog IDE okruženja.

To smo konstatovali, ali kada instalirate Python 3 na svoj računar, instalira se i jedan **interpreter**. To je tehnologija koja izvršava vaš Python kôd. To što zbunjuje je da se ovaj interpreter takođe zove „Python". U stvari, svi bi morali da koriste ispravnije ime za ovu tehnologiju, a to bi bilo „Pythonov interpreter". Ali, nažalost, niko to ne radi.

Već od ovog časa, u knjizi ćemo koristiti reč „Python" za sam jezik, a reč „interpreter" za tehnologiju koja izvršava vaš Python kôd. „IDLE" se odnosi na IDE okruženje, koje uzima vaš Python kôd i izvršava ga kroz interpreter. Ovde stvarni posao obavlja interpreter

nema GLUPIH PITANJA

P: Da li je Pythonov interpreter nešto kao Java VM?

O: I jeste i nije. Jeste, po tome što interpreter izvršava vaš kôd. Ali nije, po načinu na koji to radi. U Pythonu ne postoji postupak da se vaš izvorni kôd prevede u „izvršni". Za razliku od Jave VM, interpreter ne izvršava `.class` fajlove, nego prosto izvršava vaš kôd..

P: Ali, prevođenje ipak mora da se desi u nekoj fazi?

O: Da, dešava se, ali interpreter ne otkriva taj proces pred Python programerom (pred vama). Svi detalji se rešavaju za vas. Vi samo vidite kako se vaš kôd izvršava dok IDLE obavlja sav težak posao i komunicira sa interpreterom u vaše ime. Pričaćemo više o ovom procesu kako knjiga bude napredovala.

Izvršavanje koda, naredbu po naredbu

Ovde je ponovo programski kôd sa strane 4:

```
from datetime import datetime

odds = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
```

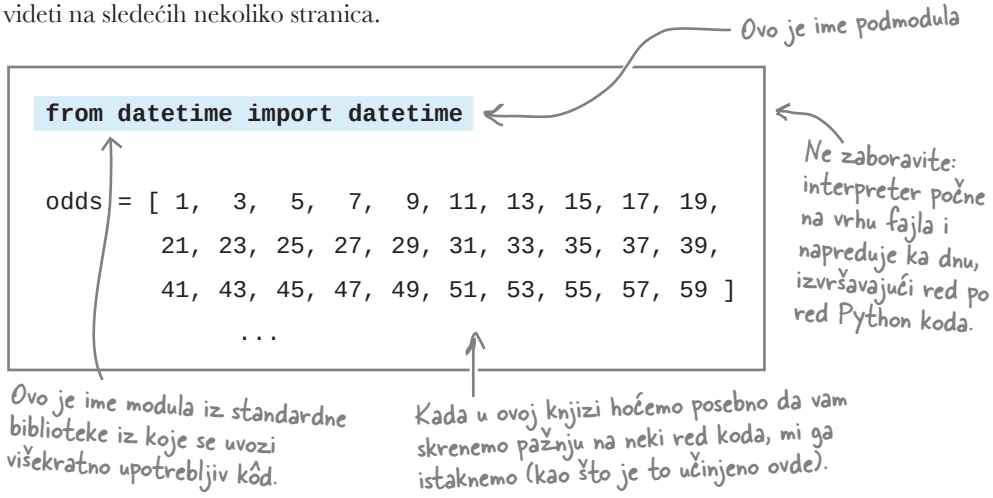
Budimo mi Pythonov interpreter

Hajde da se malo zadržimo i prođemo kroz ovaj kôd na isti način na koji to radi interpreter, red po red, od *vrha* do *dna* fajla.

Prvom linijom koda se **uvozi** (engl. *import*) neka već postojeća funkcionalnost iz Pythonove **standardne biblioteke,** a to je jedno veliko skladište softverskih modula koji sadrže mnogo unapred izrađenog (i veoma kvalitetnog) koda za višekratnu upotrebu.

U našem kodu, mi smo konkretno zatražili jedan podmodul modula `datetime` iz standardne biblioteke. Činjenica da se podmodul takođe zove `datetime` malo zbunjuje, ali to tako funkcioniše. Podmodul `datetime` obezbeđuje mehanizam kojim se saznaje vreme, kao što ćete videti na sledećih nekoliko stranica.

Module shvatite kao kolekciju povezanih funkcija.



Funkcije + Moduli = Standardna biblioteka

Pythonova **standardna biblioteka** je vrlo *bogata*, i sadrži mnogo koda za višekratnu upotrebu.

Pogledajmo još jedan modul, koji se zove `os`, koji pruža način nezavisan od platforme da se komunicira sa osnovnim operativnim sistemom (vratićemo se brzo na modul `datetime`). Koncentrišimo se samo na jednu funkciju, `getcwd`, koja – kad se pozove – vraća vaš *trenutni radni direktorijum*.

Evo kako ćete obično da uvezete (engl. *import*), a zatim pozovete (engl. *invoke*), ovu funkciju u Python programu:

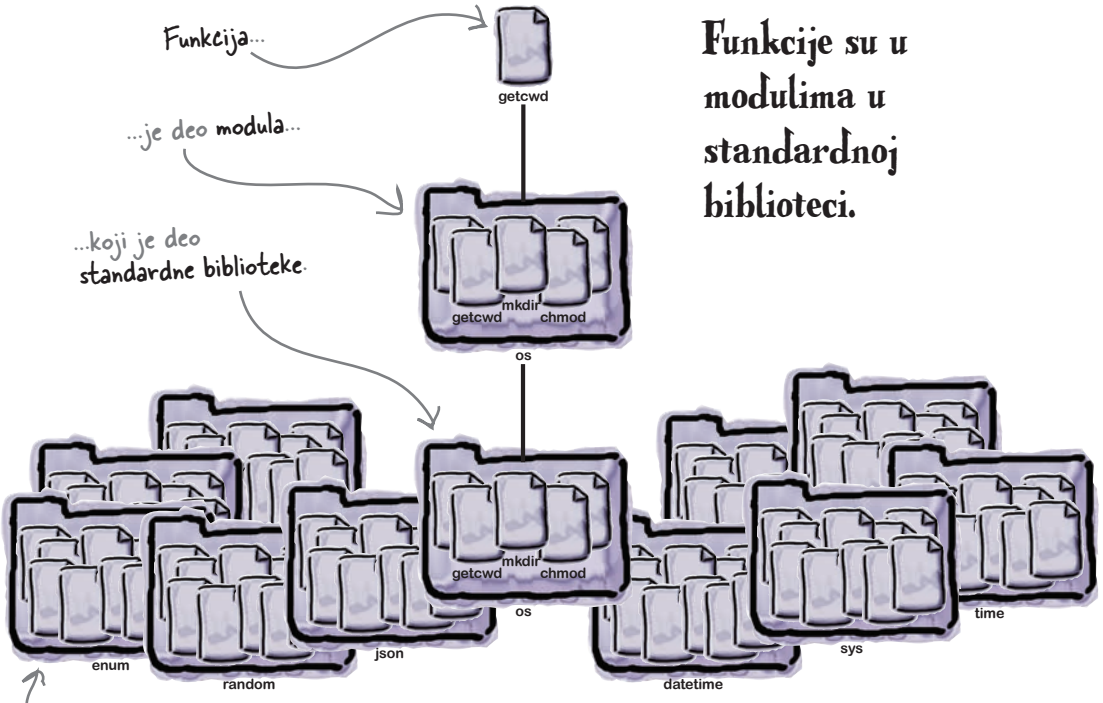
```
from os import getcwd

where_am_I = getcwd()
```

Uvezete funkciju iz njenog modula...

...a zatim je pozovete prema potrebi

Kolekcija povezanih funkcija sačinjava modul, a u standardnoj biblioteci ima *mnogo* modula:



Funkcije su u modulima u standardnoj biblioteci.

Ne brinite u ovoj fazi šta radi svaki od tih modula. Na sledećoj strani imamo kratak pregled nekih od njih, a kasnije u knjizi ćemo videti i neke od ostalih.



Standardna biblioteka izbliza

Standardna biblioteka je dragulj u Pythonovoj kruni – snabdeva vas modulima za višekratnu upotrebu koji vam pomažu u svemu od, na primer, rada sa podacima, do manipulisanja ZIP arhivama, do slanja e-pošte i rada sa HTML-om. Standardna biblioteka čak sadrži i veb server, kao i popularnu tehnologiju SQLite za baze podataka. U ovom prikazu Izbliza, predstavićemo samo nekoliko najčešće korišćenih modula iz standardne biblioteke. Da biste pratili, možete ove primere da upišete onako kako su prikazani na svoj prompt >>> (u IDLE). Ako vam je trenutno otvoren IDLE prozor za uređivanje, izaberite na meniju Run... → Python Shell da biste dobili prompt >>>.

Počnimo od malo učenja o sistemu na kojem radi vaš interpreter. Mada se Python diči time da je nezavisan od platforme, po tome što kôd pisan na jednoj platformi može da se izvrši (uglavnom neizmenjen) na drugoj, ponekad je važno da se zna da se izvršava, recimo, na sistemu Mac OS X. Modul sys postoji da bi vam pomogao da saznate više o sistemu vašeg interpretera. Evo kako ćete odrediti identitet operativnog sistema u osnovi, tako što ćete najpre da uvezete modul sys, a zatim da pristupite atributu platform:

```
>>> import sys
>>> sys.platform
'darwin'
```

Uvezite modul koji vam treba, zatim pristupite atributu koji vas zanima. Izgleda da izvršavamo „darwin", a to je ime jezgra sistema Mac OS X.

Modul sys je dobar primer modula za višekratnu upotrebu koji pre svega obezbeđuje pristup unapred postavljenim atributima (kao što je platform). Kao drugi primer, evo kako se određuje koja verzija Pythona se izvršava, što predajemo funkciji print da je prikaže na ekranu:

```
>>> print(sys.version)
3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
```

Tu je dosta informacija o verziji Pythona koju izvršavamo, uključujući to da je verzija 3.4.3.

Modul `os` je dobar primer modula za višekratnu upotrebu koji pre svega pruža funkcionalnost, a takođe i način nezavisan od sistema na koji vaš Python kôd može da komunicira sa operativnim sistemom u osnovi, bez obzira na to o kojem je tačno operativnom sistemu reč.

Na primer, evo kako da saznate ime foldera u kojem vaš kôd radi, pomoću funkcije `getcwd`. Kao sa svakim modulom, najpre uvezete modul pre nego što pozovete funkciju

```
>>> import os
>>> os.getcwd()
'/Users/HeadFirst/CodeExamples'
```

Uvezite modul, zatim pozovite funkciju koja vam treba.

Promenljivima okruženja vašeg sistema možete da pristupite kao celini (pomoću atributa `environ`) ili pojedinačno (pomoću funkcije `getenv`):

```
>>> os.environ
'environ({'XPC_FLAGS': '0x0', 'HOME': '/Users/HeadFirst', 'TMPDIR': '/var/
folders/18/t93gmhc546b7b2cngfhz10l00000gn/T/', ... 'PYTHONPATH': '/Applications/
Python 3.4/IDLE.app/Contents/Resources', ... 'SHELL': '/bin/bash', 'USER':
'HeadFirst'})'
>>> os.getenv('HOME')
'/Users/HeadFirst'
```

Atribut „environ" sadrži mnogo podataka.

Konkretnom imenovanom atributu (iz podataka sadržanih u „environ") možete da pristupite sa „getenv".

Standardna biblioteka izbliza, nastavak



Često se radi sa datumima (i vremenima), a standardna biblioteka ima modul `datetime` koji pomaže kada radite sa tom vrstom podataka. Funkcija `date.today` daje današnji datum:

```
>>> import datetime
>>> datetime.date.today()
datetime.date(2015, 5, 31)
```

Današnji datum

Ipak, to je zaista čudan način da se prikaže današnji datum, zar ne? Vrednostima dana (engl. *day*), meseca (engl. *month*) i godine (engl. year) možete da pristupite zasebno ako na poziv funkcije `date.today` dodate pristup atributu:

```
>>> datetime.date.today().day
31
>>> datetime.date.today().month
5
>>> datetime.date.today().year
2015
```

Komponente današnjeg datuma

Možete takođe da pozovete funkciju `date.isoformat` i predate joj današnji datum, da bi se prikazala mnogo razumljivija verzija današnjeg datuma, koju funkcija `isoformat` konvertuje u string:

```
>>> datetime.date.isoformat(datetime.date.today())
'2015-05-31'
```

Današnji datum u formatu string

A tu je još i vreme, kojeg izgleda niko od nas nema dovoljno. Može li standardna biblioteka da nam kaže koliko je sati? Da. Pošto uvezete modul `time`, pozovite funkciju `strftime` i navedite kako želite da se vreme prikaže. U ovom slučaju, zanima nas tekuće vreme u u satima (`%H`) i minutima (`%M`) u formatu od 24-časa:

```
>>> import time
>>> time.strftime("%H:%M")
'23:55'
```

Gospode! Zar je toliko sati?

A da utvrdimo koji je dan u nedelji i da li je pre podne ili popodne? Pomoću specifikacije `%A %p`, funkcija `strftime` upravo to radi:

```
>>> time.strftime("%A %p")
'Sunday PM'
```

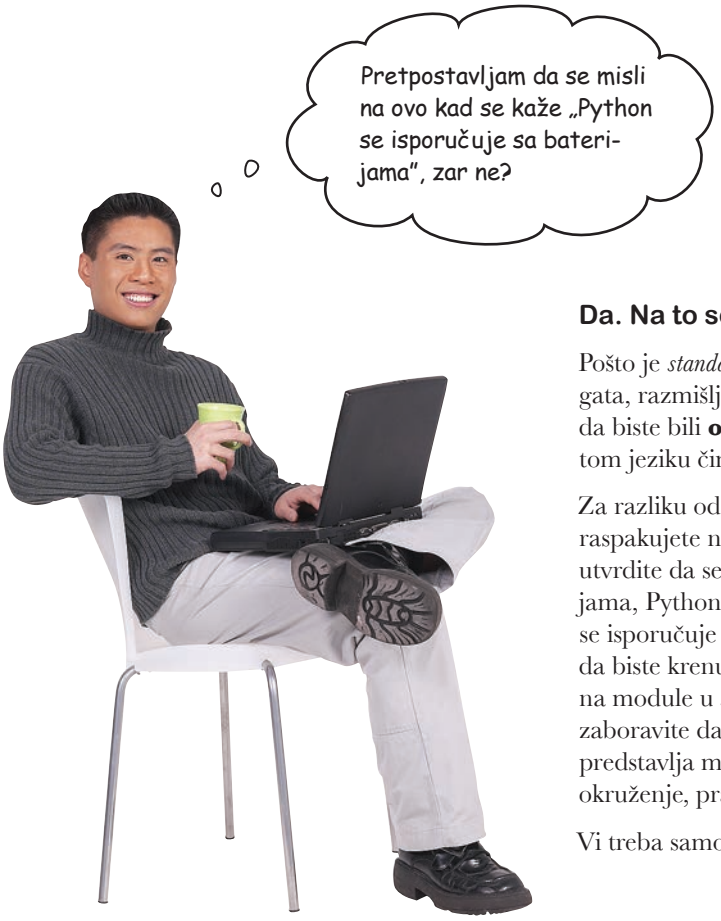
Sad smo utvrdili da je pet minuta do ponoći u nedelju uveče...vreme za spavanje, možda?

Kao poslednji primer funkcionalnosti za višekratnu upotrebu iz standardne biblioteke, zamislite da imate neki HTML a brine vas da on možda sadrži neke potencijalno opasne `<script>` oznake. Umesto da raščlanjujete HTML da biste otkrili i uklonili te oznake, zašto ne biste šifrovali sve te neugodne uglaone zagrade pomoću funkcije `escape` iz modula `html`? Ili možda imate neki šifrovani HTML koji želite da vratite u prvobitni oblik? Funkcija `unescape` može to da uradi. Ovde su primeri za oboje:

```
>>> import html
>>> html.escape("This HTML fragment contains a <script>script</script> tag.")
'This HTML fragment contains a &lt;script&gt;script&lt;/script&gt; tag.'
>>> html.unescape("I &hearts; Python's &lt;standard library&gt;.")
"I ♥ Python's <standard library>."
```

Konvertovanje u HTML šifrovani tekst i obratno

Isporučuje se sa baterijama



Da. Na to se misli.

Pošto je *standardna biblioteka toliko* bogata, razmišljanje je sve što vam treba da biste bili **odmah produktivni** u tom jeziku čim instalirate Python.

Za razliku od Božića ujutro, kad raspakujete novu igračku i razočarano utvrdite da se ne isporučuje sa baterijama, Python vas neće razočarati; on se isporučuje sa svime što je potrebno da biste krenuli. I to se ne odnosi samo na module u *standardnoj biblioteci*: ne zaboravite da je uključen IDLE, koji predstavlja malo ali upotrebljivo IDE okruženje, pravo iz kutije.

Vi treba samo da kodirate.

nema GLUPIH PITANJA

P: Kako da utvrdim šta radi neki konkretan modul iz standardne biblioteke?

O: .Python dokumentacija sadrži sve odgovore na pitanja o standardnoj biblioteci. Ovo je početna tačka: *https://docs.python.org/3/library/index.html*.



Za štrebere

Standardna biblioteka nije jedino mesto na kojem ćete naći izvrsne module koje možete da uvezete i koristite u svom kodu. Python zajednica takođe podržava jednu uspešnu kolekciju modula drugih dobavljača, od kojih ćemo neke istražiti kasnije u knjizi. Ako želite unapred da pogledate, potražite spremište koje održava zajednica: *http://pypi.python.org*.

Strukture podataka su ugrađene

Pored toga što se isporučuje sa prvoklasnom *standardnom bibliotekom*, Python ima takođe neke moćne ugrađene **strukture podataka**. Jedna od njih je **lista** (engl. list), koja može da se smatra veoma moćnim nizom (engl. *array*). Kao što se u mnogim drugim jezicima nizovi stavljaju u uglaste zagrade (`[]`), isto je sa listama u Pythonu.

Isto kao nizovi, liste mogu da drže svaku vrstu podataka.

Sledeća tri (dole prikazana) reda koda u našem programu dodeljuju unapred definisanu listu neparnih brojeva jednoj promenljivoj po imenu `odds`. U ovom kodu, `odds` je jedna *lista celih brojeva*, ali liste u Pythonu mogu da sadrže *bilo koje* podatke i to *bilo kojeg* tipa, a možete čak da mešate vrste podatka u istoj listi (ako želite). Obratite pažnju na to kako se lista `odds` širi na tri reda, i pored toga što je to jedna naredba. To je u redu, pošto interpreter neće smatrati da se naredba završila sve dok ne pronađe zatvorenu uglastu zagradu (`]`) koja je par otvorenoj (`[`). Obično, **kraj reda u Pythonu znači i kraj naredbe**, ali postoje izuzeci od tog opšteg pravila, a liste u više redova su upravo jedan od tih izuzetaka (ostale ćemo videti kasnije).

```
from datetime import datetime

odds = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]
               ...
```

Ovo je lista neparnih brojeva, obuhvaćena uglastom zagradom. To je jedna naredba koja se proteže na tri reda, a to je u redu.

Ovo je nova promenljiva po imenu „odds", kojoj se dodeljuje lista neparnih brojeva.

Sa listama može da se radi mnogo stvari, ali ćemo dalji opis odgoditi do jednog kasnijeg poglavlja. Za sada je dovoljno da znate da ova lista sada *postoji*, da je ona *dodeljena* promenljivoj `odds` (zahvaljujući primeni **operatora dodeljivanja**, `=`), i da *sadrži* prikazane brojeve.

Python promenljive se dodeljuju dinamički

Dok nismo prešli na sledeći red koda, možda je potrebno još par reči o promenljivima, pogotovo ako ste vi neki od programera koji su možda navikli na promenljive koje su unapred deklarisane sa informacijom o tipu, *pre* njihovog korišćenja (kao što je to slučaj u statički tipiziranim programskim jezicima).

Python sadrži sve uobičajene operatore, uključujući `<`, `>`, `<=`, `>=`, `==`, `!=`, a takođe i operator dodeljivanja `=`.

U Pythonu, promenljive počnu da postoje kada ih prvi put upotrebite, a **njihov tip ne mora unapred da se deklariše**. Python promenljive preuzimaju informaciju o tipu od tipa objekta koji im se dodeli. U našem programu, promenljivoj `odds` se dodeljuje jedna lista brojeva, pa je u ovom slučaju `odds` jedna lista.

Pogledajmo jednu drugu naredbu dodeljivanja promenljivoj. Srećom, to je slučajno i sledeći red koda u našem programu.

Pozivanjem metoda dobiju se rezultati

Treći red koda u našem programu je još jedna **naredba dodeljivanja**.
Za razliku od prethodnog reda, ova naredba ne dodeljuje promenljivoj jednu strukturu podataka, već jednoj drugoj novoj promenljivoj, po imenu `right_this_minute`, dodeljuje **rezultat** pozivanja jednog metoda. Pogledajte još jednom treći red koda:

Ovde se pravi još jedna promenljiva i dodeljuje joj se vrednost.

```
from datetime import datetime

odds = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
```

Ovaj poziv generiše vrednost koja će se dodeliti promenljivoj.

Pozivanje funkcionalnosti ugrađenih modula

Treći red koda poziva jedan metod po imenu `today` koji potiče iz podmodula `datetime`, a on je je *i sam* deo modula `datetime` (već smo rekli da je ova strategija imenovanja *zaista* malo zbunjujuća). Na osnovu standardnih postfiksnih zagrada: `()`se vidi da se poziva `today`.

Kasnije u knjizi će biti još ove sintakse notacije sa tačkom.

Kad se pozove `today`, on vraća „vremenski objekat", koji sadrži mnogo delova informacija o tekućem vremenu. To su **atributi** tekućeg vremena, kojima možete da pristupate uobičajenom sintaksom **notacije sa tačkom** (engl. dot-notation). U ovom programu, nas zanima atribut minuta, kojem možemo da pristupimo kada na pozivanje metoda dodamo `.minute`, kao što se vidi gore. Rezultujuća vrednost se zatim dodeljuje promenljivoj `right_this_minute`. Ovaj red koda možete da shvatite kao: *napraviti objekat koji predstavlja današnje vreme, zatim izdvojiti vrednost atributa minute pre nego što se ona dodeli promenljivoj.* Primamljivo bi bilo *podeliti* taj red koda na dva reda da bi se „lakše shvatio", na sledeći način:

Najpre, utvrditi tekuće vreme....

```
time_now = datetime.today()
right_this_minute = time_now.minute
```

...zatim izdvojiti vrednost minuta.

Možete to da uradite (ako hoćete), ali većina Python programera više voli da **ne** pravi privremene promenljive (u ovom primeru, `time_now`) *osim ako* su one potrebne negde kasnije u programu.

Odlučivanje kada da se izvrše blokovi koda

U ovoj fazi imamo listu brojeva po imenu `odds` (neparni). Takođe imamo minut po imenu `right_this_minute`. Da bismo utvrdili da li je vrednost trenutnog minuta sačuvana u `right_this_minute` neparan broj, potreban nam je način da saznamo da li se ona nalazi u listi `odds`. Ali kako da to uradimo?

Ispostavlja se da Python tu vrstu stvari rešava vrlo jednostavno. Pored toga što sadrži sve uobičajene operatore poređenja koje očekujete u svakom programskom jeziku (kao što su `>`, `<`, `>=`, `<=`, i tako dalje), Python ima i nekoliko vlastitih „super" operatora, od kojih je jedan `in`.

Operator `in` ispituje da li je jedna stvar *unutar* (engl. *inside*) druge. Pogledajte sledeći red koda u našem programu, gde se operator `in` koristi za proveravanje da li se `right_this_minute` nalazi *unutar* liste `odds`:

```
        ...
right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
        ...
```

Operator „in" je moćan. On može da utvrdi da li je jedna stvar unutar druge.

Ova naredba „if" može da ispadne bilo „True", bilo „False".

Operator `in` može da vrati `True` (tačno) ili `False` (netačno). Kao što biste očekivali, ako se vrednost `right_this_minute` nalazi u `odds`, naredba `if` ispada `True`, pa će se izvršiti blok koda pridružen naredbi `if`.

Blokovi se u Pythonu lako uočavaju, pošto su uvek uvučeni.

U našem programu postoje dva bloka, od kojih svaki sadrži samo jedan poziv funkcije `print`. Ta funkcija može da prikaže poruke na ekranu (i videćemo da se često koristi u celoj ovoj knjizi). Kad ste upisivali ovaj programski kôd u prozor za uređivanje, možda ste primetili da vam IDLE pomaže da ne pogrešite tako što vrši automatsko uvlačenje. To je vrlo korisno, ali obratite pažnju na to da li je IDLE izvršio uvlačenje baš kao što ste vi hteli:

```
        ...
right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
```

Funkcija „print" prikazuje poruku na standardnom izlazu (tj., na vašem ekranu).

Ovde je jedan blok koda. Napomena: kôd je uvučen.

A ovde je još jedan blok koda. Napomena: i on je uvučen.

Jeste li primetili da ovde nema vitičastih zagrada?

Šta je bilo sa vitičastim zagradama?

Ako ste navikli na programski jezik u kojem se za razgraničavanje blokova koda koriste vitičaste zagrade (`{` i `}`), prvi susret sa blokovima u Pythonu može da vas zbuni, pošto Python ne koristi vitičaste zagrade u tu svrhu. Python koristi uvlačenje za razgraničavanje blokova koda, koje Python programeri nazivaju **svita** (engl. *suite*) a ne *blok* (samo da bi malo zamutili stvari).

Umesto da kažu „blok" koda, Python programeri koriste reč „svita". U praksi se koriste oba naziva, ali stručnjaci za Python više vole „svita".

Nije stvar u tome da se vitičaste zagrade ne koriste u Pythonu. Koriste se, ali – kao što ćemo videti u poglavlju 3 – vitičaste zagrade se koriste za razgraničavanje podataka, a ne za razgraničavanje sviti (tj., *blokova*) koda.

Svite se u svakom Python programu lako uočavaju, pošto su ti blokovi uvek uvučeni. To vam pomaže da ih brzo prepoznate kada čitate kôd.

Drugi vizuelni znak raspoznavanja koji treba da primetite je znak dvotačka (`:`), koji se koristi kao uvod u svitu pridruženu nekoj od Pythonovih kontrolnih naredbi (kao što su `if`, `else`, `for`, i slične). Videćete mnoge takve primere dok budete napredovali kroz ovu knjigu.

Dvotačka uv odi uvučenu svitu koda

Dvotačka (`:`) je važna, po tome što uvodi novu svitu koda koji mora da bude uvučen na desnu stranu. Ako ste zaboravili da uvučete kôd iza dvotačke, interpreter prijavljuje grešku.

Ne samo da naredba `if` u našem primeru ima dvotačku, ima je i naredba `else`. Ovde je ponovo ceo kôd:

```
from datetime import datetime

odds = [ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19,
        21, 23, 25, 27, 29, 31, 33, 35, 37, 39,
        41, 43, 45, 47, 49, 51, 53, 55, 57, 59 ]

right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
```

Dvotačke uvode uvučene svite.

Skoro smo gotovi. Ostaje nam da opišemo samo još jednu naredbu.

Kakav „else" može da bude uz „if"?

Skoro smo završili sa kodom našeg programa za primer, jer je preostao da se opiše samo jedan red koda. To nije mnogo veliki red koda, ali je važan: naredba `else` koja uvodi blok koda koji se izvršava kada odgovarajuća naredba `if` vrati vrednost `False`.

Pogledajte bolje naredbu `else` u našem programskom kodu, koja nije uvučena da bi bi bila poravnata sa `if` delom ove naredbe:

```
if right_this_minute in odds:
    print("This minute seems a little odd.")
else:
    print("Not an odd minute.")
```

Vidite dvotačku?

Jeste li primetili da „else" nije uvučen, da bi bio poravnat sa „if"?



Pretpostavljam da ako postoji „else", mora da postoji i „else if", ili se u Pythonu piše „elseif"?

Veoma je česta greška da početnici u Pythonu zaborave dvotačku kad prvi put pišu kôd.

Ni jedno ni drugo. U Pythonu se piše `elif`.

Ako imate niz uslova koje proveravate u okviru `if` naredbe, Python ima `elif`,a ne samo `else`. Možete da imate onoliko `elif` naredbi (svaku sa svojom svitom) koliko vam god treba.

Ovde je mali primer u kojem se polazi od toga da je promenljivoj po imenu `today` prethodno dodeljena string vrednost naziva današnjeg dana (Ako je subota: zabava, inače ako je nedelja: oporavak, inače: rad, rad, rad):

```
if today == 'Saturday':
    print('Party!!')
elif today == 'Sunday':
    print('Recover.')
else:
    print('Work, work, work.')
```

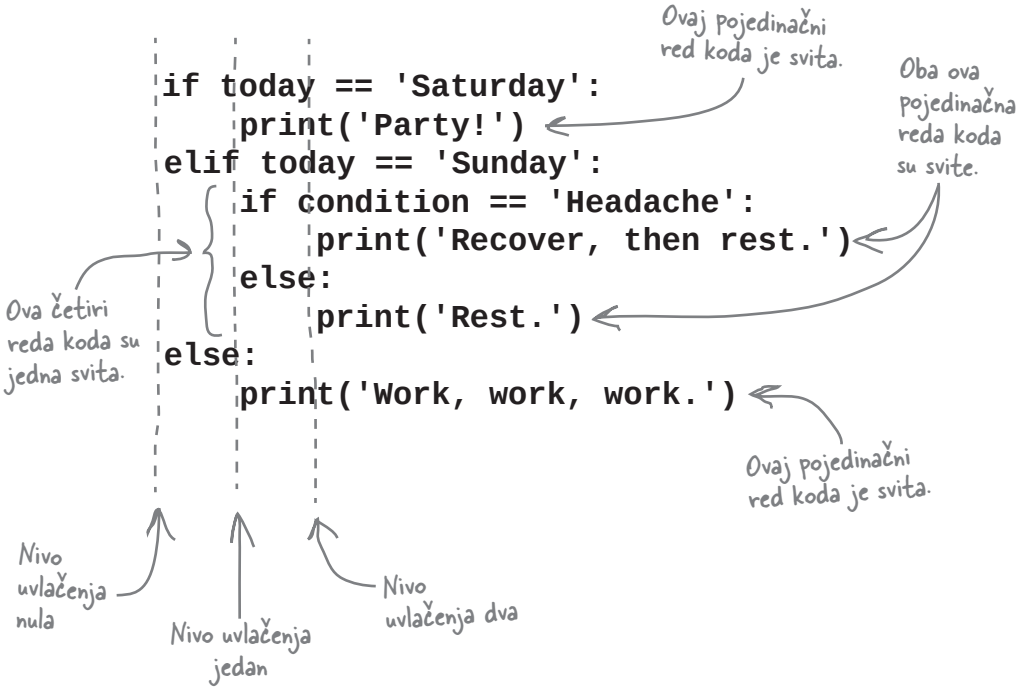
Tri pojedinačne svite: jedna za „if", druga za „elif", i konačna koja hvata sve ostalo, za „else".

Svite mogu da sadrže ugnežđene svite

Svaka svita može da sadrži bilo koji broj ugnežđenih svita, koje takođe moraju da budu uvučene. Kada Python programeri pominju ugneždene svite, obično pominju i **nivoe uvlačenja**.

Početni nivo uvlačenja bilo kojeg programa obično se naziva *prvi* ili (kao što je u mnogim programskim jezicima uobičajeno da se broji) nivo uvlačenja *nula*. Naredni nivoi se nazivaju drugi, treći, četvrti, i tako dalje (ili nivo jedan, nivo dva, nivo tri, i tako dalje).

Ovde je jedna varijacija primera koda za današnji dan sa prethodne strane. Primetićete kako je ugnežđeno `if/else` dodato u naredbu `if` koja se izvršava kada `today` ima vrednost `'Sunday'`. Mi takođe pretpostavljamo da postoji još jedna promenljiva po imenu `condition` i da joj je dodeljena vrednost koja izražava kako se trenutno osećate. Pokazali smo gde je svaka svita, kao i nivo uvlačenja na kojem se nalazi:




```
if today == 'Saturday':
    print('Party!')
elif today == 'Sunday':
    if condition == 'Headache':
        print('Recover, then rest.')
    else:
        print('Rest.')
else:
    print('Work, work, work.')
```

Važno je da se primeti da je kôd na istom nivou uvlačenja povezan sa ostalim kodom na istom nivou uvlačenja jedino ako se sav taj kod nalazi *unutar iste svite*. Inače, oni su u zasebnim svitama, pa nije važno to što su na istom nivou uvlačenja. Bitno je da se u Pythonu uvlačenje koristi da bi se razgraničile svite koda.

Šta već znamo

Pošto smo opisali poslednjih nekoliko redova koda, zastanimo malo da rekapituliramo šta nas je program `odd.py` naučio o Pythonu:

UKRATKO

- Python se isporučuje sa ugrađenim IDE okruženjem po imenu IDLE, koje vam omogućava da pravite, menjate i izvršavate svoj Python kôd – dovoljno je da upišete kôd, sačuvate ga i zatim pritisnete F5.
- IDLE komunicira sa Pythonovim interpreterom, koji vam automatizuje proces prevod-povezivanje-izvršavanje. To vam omogućava da se koncentrišete na pisanje koda.
- Interpreter izvršava vaš kôd (sačuvan u fajlu) od vrha ka dnu, red po red. U Pythonu ne postoji pojam funkcije/metoda `main()`.
- Python se isporučuje sa moćnom standardnom bibliotekom, koja obezbeđuje pristup mnogim modulima za višekratnu upotrebu (od kojih je `datetime` samo jedan primer).
- Kada pišete Python programe imate na raspolaganju celu kolekciju standardnih struktura podataka. Lista je jedna od njih, a to je pojam veoma sličan nizu.
- Tip promenljive ne mora da se deklariše. Kada u Pythonu promenljivoj dodelite vrednost, ona dinamički preuzima tip podataka na koje se odnosi.
- Odluke donosite pomoću naredbe `if`/`elif`/`else`. Ključne reči `if`, `elif`, i `else` prethode blokovima koda, koji se u Pythonu zovu „svite”.
- Svite koda se lako uočavaju, pošto su uvek uvučene. Uvlačenje je jedini mehanizam za grupisanje koda u Pythonu.
- Pored uvlačenja, svitama koda takođe prethodi dvotačka (:). To je sintaksni zahtev jezika



Hajde da proširimo program da radi još više.

Istina je da smo utrošili više redova da bismo opisali šta ovaj kratak program radi nego što je bilo potrebno da se napiše sam kôd. Ali to je jedna od velikih prednosti Pythona: *možete mnogo da uradite sa svega nekoliko redova koda.*

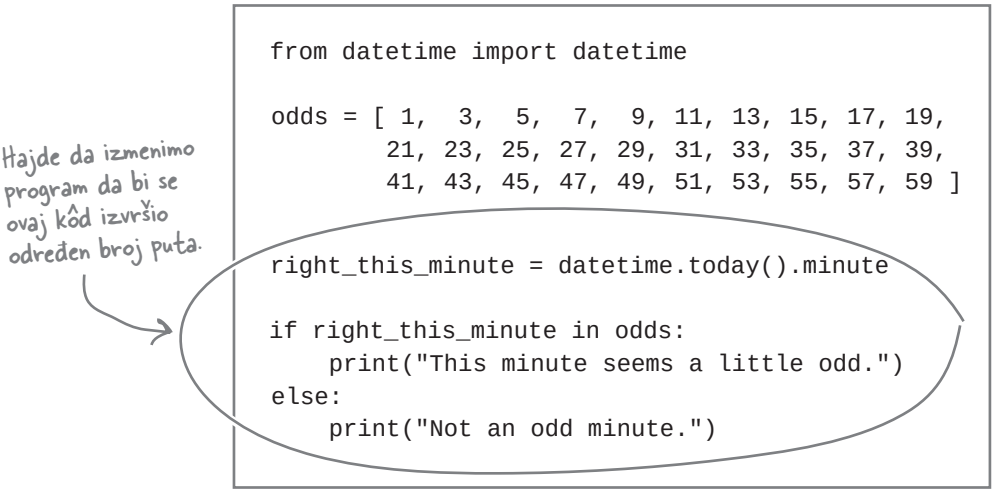
Pregledajte još jednom gornju listu, a zatim okrenite stranicu, pa ćete videti šta će biti proširenja našeg programa.

Proširimo program da radi više

Hajde da proširimo naš program da bismo naučili još nešto o Pythonu.

Trenutno, program se izvrši jednom i zatim se završava. Zamislite da hoćemo da se ovaj program izvrši više puta; recimo, pet puta. Konkretno, hajde da izvršimo „kôd za ispitivanje minuta" i naredbu `if/else` pet puta, sa pauzom od slučajnog broja sekundi između prikazivanja poruka (da sve bude malo zanimljivije). Kada se program završi, na ekranu bi trebalo da stoji pet poruka, umesto ove jedne.

Ovde je ponovo sadašnji kôd, a zaokružen je deo koji hoćemo da izvršimo više puta:



Šta treba da uradimo:

1. **Napravimo petlju oko zaokruženog koda.**
Petlja omogućava iteraciju nad bilo kojom svitom, a Python predviđa niz načina da se to uradi. U ovom primeru (nećemo objašnjavati zašto), upotrebićemo za ponavljanje Pythonovu petlju `for`.

2. **Pauza u izvršavanju.**
Pythonov standardni modul `time` sadrži jednu funkciju po imenu `sleep` koja može da pauzira izvršavanje za zadati broj sekundi.

3. **Generišemo slučajan broj.**
Srećom, još jedan Pythonov modul, `random`, nudi funkciju po imenu `randint` koju možemo da upotrebimo za generisanje slučajnog broja. Upotrebićemo `randint` da nam napravi broj između 1 i 60, a zatim ćemo upotrebiti taj broj da odredimo pauzu u izvršavanju našeg programa pri svakom ponavljanju.

Sada znamo šta hoćemo da uradimo. Ali koji će biti najbolji način da se izvrše ove izmene?