Обектно ориентирано програмиране

2-ро издание
Версия Python 2.3

Да научим

Python

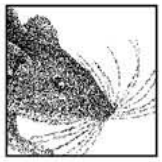
O'REILLY®

ЗеСТ Пресс

Марк Луц
и Дейвид Ашер

O'REILLY®

Да научим Python



Python е популярен език за обектно ориентирано програмиране с отворен сорс, използван както за създаването на самостоятелни програми, така и на скриптови приложения. Преносим, мощен и лесен за употреба, а за овладяването му възползвайте от помощта на експертните учители. Второто издание на *Да научим Python* ви оставя в ръцете на Марк Луц и Дейвид Ашер, двама именити експерти относно Python, чийто „дружелюбен“ и добре структуриран текст е помогнал на голям брой програмисти да достигнат майсторство във владеенето на езика.

Освен самите характеристики на езика, това издание на *Да научим Python* предлага също нов контекст за не толкова опитните програмист: нов общ преглед на обектно ориентираното програмиране, а също и на динамичния стил на програмиране, нови дискусии за начините на стартиране на програмите, както и на възможностите за конфигуриране, представени са по нов начин източниците за документация и др. В книгата са разгледани и нови случаи от практиката, така че да бъдат показани по-конкретно приложението на възможностите на езика.

За начало *Да научим Python* предлага на програмистите цялата необходима информация за разбирането и изграждането на програми с езика Python – за типовете, операторите, конструкциите, класовете, функциите, модулите и изключенията. След което авторите представят по-задълбочен материал, показвайки как с Python могат да се изпълняват често срещани в практиката задачи, като се използват съществуващи приложения и библиотеки към тези приложения. Книгата съдържа също и множество упражнения, с които можете да проверите новопридобитите умения.

Да научим Python, второ издание, е самостоятелна книга, позволяваща на читателите да навлязат в дълбочина в сърцевината на самия език Python. С прочита на книгата ще постигнете цялостно и задълбочено разбиране на Python, което ще ви помогне сами да кодирате по-големи приложения. Тази книга е предназначена за всички, които не просто искат да научат Python, а най-вече майсторски да го овладеят.

Коментар към първото издание:

> ***„Една от най-положителните страни на книгата е, че авторите са постигнали почти съвършен баланс между полезните примери и изчерпателните описания. Независимо дали сте опитен програмист, или напълно начинаещ потребител, тази книга е идеална за изучаването на езика Python“.***
>
> —Andrew Morrison, CedarLug

ISBN 954-9341-06-2

9 789549 341065 23

Да научим Python

Марк Луц и Дейвид Ашер

Да научим Python

превод
Валентин Веселинов Димитров

O'REILLY®

София
2006

ЗеСТ Прес

Learning Python, Second Edition

by Mark Lutz and David Ascher

Translator: Valentin Veselinov Dimitrov
Scientific editor: Hristo Nikolov
Production editor and copy editor: Valya Tontcheva
Printing-press: "INVESTPRESS"

Bulgarian language edition published by ZeST Press Publishing House, Ltd, 2006
ISBN -10: 954-9341-14-3
ISBN -13: 978-954-9341-14-0

Да научим Python, второ издание

от Марк Луц и Дейвид Ашер

Превод: Валентин Веселинов Димитров
Научен редактор: Христо Николов
Редактор и коректор: Валя Тончева
Печат: "ИНВЕСТПРЕС", АД

Издава на български език: Издателска къща "ЗеСТ Прес", ЕООД, 2006
ISBN -10: 954-9341-14-3
ISBN -13: 978-954-9341-14-0

Съдържание

Част II. Типове и операции

Част III. Конструкции и синтаксис

Част IV. Функции

Част V. Модули

Част VI. Класове и ООП

Част VIII. Външните слоеве

Част IX. Приложения

Индекс

Предговор

Тази книга ви предлага въведение към езика за програмиране Python. Python е популярен обектно ориентиран език, който се използва за създаването както на самостоятелни програми, така и на контролиращи приложения в най-различни области. Той е безплатен, преносим, мощен и изключително лесен за работа.

Независимо дали сте начинаещ програмист или опитен разработчик, целта на книгата, която държите в ръцете си, е да ви помогне бързо да навлезете в същността на езика Python.

Относно второто издание

През четирите години, които изминаха след публикуването на първото издание на книгата в края на 1998 година, настъпиха значителни промени както в самия език Python, така и в проблематиката, представяна от авторите по време на водените от тях работни групи по изучаване на Python. Въпреки усилията ни да запазим колкото е възможно по-голяма част от първоначалното издание, това ново издание отразява съвременните тенденции в преподаването на Python, както и по-новите промени в езика.

По отношение на самия език тази книга е изцяло осъвременена, за да отразява версия 2.2 на Python, както и настъпилите промени след излизането на първото издание от печат. Вмъкната е също и дискусия за очакваните промени в предстоящата версия 2.3. Вижте някои от основните въпроси, свързани с езика, за които ще откриете нова или допълнена информация:

- Включване на списъци (глава 14)
- Класови изключения (глава 25)
- Методи за знакови низове (глава 5)
- Подсилено задаване на стойност (глава 8)
- Традиционно, истинско и целочислено деление (глава 4)
- Пакетно импортиране (глава 17)
- Обхвати на вложените функции (глава 13)

- Генераторни функции и итератори (глава 14)
- Unicode знакови низове (глава 5)
- Разделяне на типовете на подкласове (глава 7 и глава 23)
- Статични методи и методи за класове (глава 23)
- Псевдоизолирани атрибути на класове (глава 23)
- Разширени конструкции `print` и `import` (глава 8 и глава 18)
- Нови вградени инструменти, например `zip` и `isinstance` (глава 7 и глава 10)
- Класове от нов вид (глава 23)
- Нови възможности за стартиране на програмите и настройка, файловете с разширение *pth* (глава 3 и приложение А)
- Нови инструменти за разработка, например IDLE, Psyco, Py2exe и Installer (глава 2, глава 3 и глава 29)
- Нови инструменти за документиране и тестване на кода, например PyDoc, PyUnit и doctest (глава 11)

В процеса на четенето на книгата ще се запознаете и с по-незначителни промени в езика (например насърчаване на използването на дългите цели числа, списъци за експортиране на модули). Освен посочените промени, в това издание сме разширили частта, посветена на основното ядро на езика (част I – част VII) с допълнителен материал и примери, представяни в уроците за изучаване на Python в работните групи, водени от Mark през последните няколко години. Така например ще откриете:

- Ново въведение в ООП (обектно ориентираното програмиране) (глава 19)
- Нов обзор на динамичното програмиране (глава 4)
- Нов преглед на инструментите за разработка (глава 26)
- Допълнителен материал относно архитектурата и изпълнението на програмите (глава 2, глава 3 и глава 15)
- Нов преглед на източниците на документация (глава 11)

Направихме много допълнения и промени в частта, посветена на основата на самия език, с цел да улесним начинаещите потребители. Ще видите също, че много от въпросите, свързани с ядрото на езика, в това издание са значително разширени, като са добавени множество разисквания и примери. Тъй като този текст в голяма степен се превърна в основен източник за желаещите да изучат основата на езика Python, позволихме си да разгледаме проблема много по-задълбочено и изчерпателно, като междувременно добавихме и нови аспекти. По същата логика осъвременихме и част VIII, така че да отразява по-новите области на приложение, както и модерните модели на програмиране.

Нещо повече, изданието, което държите в ръцете си, включва нов набор от съвети и „трикове" при използването на Python, почерпани както от групите за изучаване, които водихме през последните седем години, така и от опита при използването на Python за реализирането на действителни задачи през последното десетилетие. Упражненията са разширени и осъвременени, така че да отразят модерната практика при използването на Python, новите възможности на езика, както и по-често срещаните грешки, допускани от начинаещите потребители, които наблюдавахме „от първа ръка" през последните години. Изобщо това издание е по-обемисто на първо място, тъй като самият Python се е разширил, а също понеже сме добавили контекст, доказал значимостта си в практиката.

Съобразявайки се с факта, че това издание е по-изчерпателно, разделихме по-голямата част от първоначалните глави на по-лесно „смилаеми" части. Тоест реорганизирахме частта, посветена на основата на езика, в няколко раздела, включващи по няколко глави, за да улесним възприемането на целия материал. Така например типовете и конструкциите съставляват две основни части, като на всеки по-важен проблем, свързан с тях, е посветена отделна глава.

Целта на тази нова подредба е да ни позволи да кажем повече, без да затормозяваме читателите. Междувременно преместихме упражненията и „уловките" от края на всяка от главите в края на всяка включваща ги част, така че те вече се намират в края на последната глава на всяка част.

Въпреки добавената нова проблематика, тази книга е ориентирана към начинаещите потребители на Python, като нейното предназначение е да бъде първа стъпка за програмистите, желаещи да изучат Python.[1] Запазили сме значителна част от материала, структурата и насоката на първото издание. На местата, където това е възможно, сме разширили въведенията за начинаещите потребители, като сме изместили по-сложните нови въпроси от основната част на материала, с цел да избегнем ненужното усложняване на фундаменталната проблематика. Нещо повече, тъй като е базирано върху издържал проверката на времето опит в преподаването на учебния материал, това издание (също както първото) може да ви послужи като самоучител за Python.

Необходими познания

Всъщност предварителните познания не са безусловно необходими. Тази книга е използвана успешно както от напълно начинаещи, така и от "печени" ветерани програмисти. Но стигнахме и до наблюдението, че всякакъв предишен опит с програми или скриптове ще ви бъде от полза, макар това да не е задължително за всеки читател.

Предназначението на книгата е да бъде въвеждащ в Python текст за програмисти. Тя може би няма да е най-доброто учебно помагало за човек, който не се е докосвал до компютър (няма да обясняваме какво представлява компютърът), но от друга страна не очакваме от читателите опит в програмирането или специално образование.

[1] Под „програмист" разбираме всеки, който е написал поне един ред код, на който и да е програмен или скриптов език. Ако не отговаряте на това условие, тази книга пак ще ви бъде полезна. Но тук отделяме повече внимание на преподаването на самия Python, отколкото на основните принципи на програмирането.

При все това няма да обиждаме читателите, като приемем, че са „dummies” (глупчовци, мухльовци), каквото и да означава тази дума; с Python извършването на различни полезни неща става изключително лесно и се надяваме да ви покажем как и вие можете да направите това. На места в текста съпоставяме Python с езици за програмиране от рода на C, C++, Java и Pascal, но съвсем спокойно можете да пренебрегнете тези сравнения, ако не сте използвали някои от тях преди това.

Може би от самото начало трябва да спомена следното: създателят на Python, Guido van Rossum, му е дал име от сериала на BBC *Monty Python's Flying Circus* (*Летящият цирк на Monty Python*). Това наследство неизменно прибавя комичен привкус на някои от примерите в Python. Така например в света на Python, както и в част от кода, който ще видите в книгата, традиционните „foo” и „bar” (метасинтактични променливи, обикновено използвани в програмирането) са заместени от „spam” (кълцана шунка) and „eggs” (яйца). Поради същата причина можете да срещнете и имената на „Brian,” „Ni” и „shrubbery” (градински храсти). Не е необходимо да сте запознати със сериала, за да разберете примерите (символите са просто символи), но ако сте го гледали, няма да ви навреди.

Обхват на книгата

Въпреки че тази книга представя всички основни аспекти на езика Python, обхватът й е стеснен, с цел да намалим времето, необходимо за нейния прочит, както и размера й. В текста обръщаме по-голямо внимание на основните понятия, използваме малки и самостоятелни примери, за да онагледим казаното, като на места пропускаме по-незначителните детайли, които лесно бихте могли да прочетете в съществуващите наръчници. Поради това може би най-добрият начин да опишем мястото на тази книга е като въведение и едновременно с това крайъгълен камък по пътя към по-задълбочените и изчерпателни текстове.

Така например няма да навлизаме в подробности по отношение на интеграцията на Python и C – сложен въпрос, който въпреки това стои в основата на разработката на множество базиращи се на Python системи. Няма да се задълбочаваме също и по отношение на историята и процеса на еволюцията на Python. Някои популярни приложения за Python, например графични потребителски интерфейси, системни инструменти, както и предназначените за мрежови контрол, са разгледани съвсем накратко. Съвсем естествено този обхват пропуска част от цялостната картина.

Като цяло Python вдига с няколко степени равнището на качеството в света на скриптовите езици. Част от свързаните с Python концепции се нуждаят от по-обширен контекст, отколкото тук сме в състояние да ви предложим, и би било проява на небрежност от наша страна да не ви препоръчаме да продължите с обучението си и след като приключите с четенето на книгата. Надяваме се, че по-голямата част от нашите читатели ще се насочат и към допълнителни текстове, за да придобият по-пълни познания относно програмирането на приложения.

Поради насочеността си предимно към начинаещите потребители, *Да научим Python* съвсем естествено се допълва от останалите книги на издателство O'Reilly, посветени на Python. Така например *Програмиране с Python*, Второ издание предлага по-обширни и по-задълбочени примери с приложения и е преднамерено замислена като продължение на текста, който четете в момента. Изобщо изданията на *Да научим Python* и *Програмиране с Python* отразяват двете части от материала, преподаван от Mark – основната част на езика и програмирането на приложения. *Джобен справочник за Python*, Второ издание, на O'Reilly може да ви послужи като допълнителен справочник, в който можете да откриете по-деликатните подробности, които ще пропуснем тук.

Можете да намерите допълнителни примери с код в други посветени на Python книги, които да използвате като допълнителен източник на сведения, а също и за да изследвате по-специфични области при използването на Python. Като източник на сведения и основен справочник ви препоръчваме *Python: накратко* на O'Reilly, както и *Основен справочник за Python* на New Riders, а като библиотека с примери – *Python Cookbook* на O'Reilly. Независимо от това коя книга ще изберете, трябва да осъзнаете, че останалата част от цялостната картина на Python изисква да изследвате примери, които са по-реалистични от тези, за които можем да си позволим да отделим място тук. Днес има близо 40 книги за Python, които можете да намерите на английски език, а също и няколко десетки издания на други езици. Тъй като четенето е субективно изживяване, съветваме ви да прегледате всяка от наличните книги, за да откриете тази, която най-добре подхожда на вашия вкус и интерес.

Но въпреки ограничения обхват (а може би именно поради него), считаме, че тази книга ще бъде добро начало за изучаване на Python. Ще научите всичко необходимо, за да можете да започнете да кодирате самостоятелни програми и скриптове с Python. След като приключите с четенето на книгата, ще сте изучили не само езика Python, но и как да го използвате при извършването на ежедневни задачи. Ще бъдете подготвени да се справите с по-сложните проблеми и примери, когато се изпречат на пътя ви.

Стил и структура на книгата

По-голямата част от книгата е базирана върху учебен материал, предназначен за тридневен практически курс за изучаване на Python. Накрая на всяка от главите в частите, посветени на основата на езика, ще откриете упражнения, отговорите на които се намират в приложение B. Целта на упражненията е да ви позволят да започнете да кодирате от самото начало, което е и една от най-привлекателните страни на предложения курс.

Препоръчваме ви да правите упражненията успоредно с четенето, не само за да добиете опит с програмирането на Python, а също понеже в някои от тях се разглеждат въпроси, които не се обсъждат на друго място в книгата. Ако срещнете затруднение, отговорите и решенията в приложение B ще ви бъдат от помощ (като ви съветваме да „хитрувате", колкото ви харесва). Съвсем естествено, за да можете да работите с упражненията, ще трябва да инсталирате Python.

Тъй като целта на текста е бързо да представи основните аспекти на езика, организирали сме изложението според основните характеристики на Python, а не според примерите. Подходът ни при написването на книгата е постепенно да надграждаме получените от читателя знания: от вградените обектни типове, през конструкциите до елементите на програмите и т.н. Всяка от главите е до голяма степен самостоятелна, но в по-късните глави използваме концепции, представени в по-раншните (така например, когато стигнете до класовете, приемаме, че вече знаете как да кодирате функции), така че може би ще бъде най-добре да четете книгата последователно. От по-обща гледна точка книгата е разделена на няколко функционални области и на съответстващите им части.

Основна част на езика

Този раздел на книгата представя езика Python по споменатия начин на постепенно надграждане. Организирали сме раздела, така че на всяка основна характеристика на езика съответства по една част – типове, функции и т.н. – като по-голямата част от примерите са малки и самостоятелни (може би някои читатели биха възразили, че примерите в книгата са до известна степен изкуствени, но те илюстрират успешно това, което искаме да ви покажем). Този раздел представлява основната част от текста, което е достатъчно красноречиво за замисъла на книгата. Той е съставен от следните части:

Част I, *Първи стъпки*

Започваме с общ преглед на Python, в който даваме отговор на някои често задавани от начинаещите потребители въпроси – защо хората го предпочитат като език за програмиране, какво може да се направи с него и т.н. В първата глава представяме няколко по-важни идеи, върху които се базира неговата технология, за да ви помогнем да придобиете известни основни познания.

След което постепенно навлизаме в техническата материя, разглеждайки начините, по които вие и Python стартирате програмите. Целта ни тук е да ви дадем достатъчно познания, за да можете да работите успоредно с упражненията, показани по-нататък в книгата. Ако имате нужда от допълнителна помощ, за да започнете работа с Python, в приложение А ще откриете повече подробности относно начините за неговото конфигуриране.

Част II, *Типове и операции*

В следващата част започва разходката ни из езика Python, в нея изследваме по-задълбочено основните вградени обектни типове: числа, списъци, речници и т.н. Бихте могли да свършите много неща с Python и само с помощта на тези инструменти.

Част III, *Конструкции и синтаксис*

В следващата част въвеждаме конструкциите на Python – кодът, който пишете, за да създадете и обработите неговите обекти. Тук представяме и общия модел на синтаксиса на Python.

Част IV, *Функции*

С тази част започваме прегледа на инструментите на Python за изграждане на програмни структури от по-високо ниво. Функциите се оказват прост начин за пакетиране на кода с цел повторната му употреба.

Част V, *Модули*

Модулите на Python ви позволяват да организирате конструкциите и функциите в по-големи компоненти, като в тази част ви показваме как да създавате, използвате, импортирате и зареждате повторно модули.

Част VI, *Класове и ООП*

Тук изследваме инструмента на Python за обектно ориентирано програмиране (ООП), класовете. Както сами ще видите, в Python ООП е свързано основно с търсенето на имена в свързани обекти.

Част VII, *Изключения и инструменти*

Завършваме раздела, посветен на основната част на езика, с преглед на конструкциите и модела за манипулиране на изключенията в Python, както и с кратък обзор на инструментите за разработка. Тази част е последна, понеже изключенията е възможно да бъдат и класове, ако вие желаете това.

Външните слоеве

В част VIII отново представяме вградените инструменти на Python, като ги използваме в няколко малки примерни програми. Показваме ви също как да извършите някои от по-често срещаните задачи за кодиране с помощта на Python, така че да придобиете известен практически усет, за използването както на самия език, така и на стандартните му библиотеки и инструменти.

Глава 27, *Често срещани задачи в Python*

В тази глава ви представяме подбрани модули и функции, включени при стандартната инсталация на Python. По дефиниция те представляват минималният набор от модули, до които всеки потребител на Python би трябвало да има достъп. Познаването на възможностите на стандартния комплект от инструменти вероятно ще ви спести седмици работа.

Глава 28, *Работни рамки*

Тук ще ви покажем няколко действителни приложения. Базирайки се на ядрото на езика, разгледано в предишните части от книгата, а също на вградените инструменти, описани в глава 27, в тази глава ви представяме няколко малки, но полезни програми, които показват как да използвате натрупаните дотук знания. Разглеждаме три области, които представляват интерес за по-голямата част от потребителите на Python: по-основни задачи, обработка на текстове и системни интерфейси. Завършваме тази глава с преглед на Jython, базираната на Java версия на Python, както и с една по-обемиста програма, написана на Jython.

Глава 29, *Ресурси за Python*

В тази глава обсъждаме слоевете на общността на Python, както и няколко специализирани библиотеки, които или представляват част от стандартния софтуерен пакет на Python, или са независимо разработени от трети страни и се разпространяват безплатно.

Приложения

Приключваме книгата с две приложения, в които даваме няколко съвета за използването на Python под различни платформи (приложение А), а също и отговори на упражненията, намиращи се в края на последната глава на всяка част (приложение Б). Обърнете внимание, че можете да използвате индекса и съдържанието, за да търсите по-конкретни неща, но в настоящата книга няма справочни приложения. Както споменахме по-рано, *Джобният справочник за Python*, Второ издание (O'Reilly), заедно с други книги, а също и безплатните наръчници за Python, които можете да откриете в Интернет на адрес *http://www.python.org*, допълват подробностите относно синтаксиса и вградените инструменти на Python.

Актуализация на книгата

С течение на времето се извършват подобрения (а стават също и ^H^H^H печатни грешки). Ще откриете актуализирана информация, допълнения и корекции на книгата в уеб на един от следващите сайтове:

- *http://www.oreilly.com (сайт на O'Reilly)*
- *http://www.rmi.net/~lutz (сайт на Mark)*
- *http://starship.python.net/~da (сайт на David)*
- *http://www.python.org (начален уебсайт на Python)*
- *http://www.rmi.net/~lutz/about-lp.html (уеб страница на книгата)*

Ако това е възможно, бихме били по-конкретни, но уеб се променя по-бързо от отпечатаните книги.

Уговорки относно шрифтовете

В тази книга използваме следните уговорки при форматирането на текста:

Курсивен шрифт
: За адреси на електронна поща, имена на файлове, адреси на ресурси в Интернет (URL), за да наблегнем на новите термини, когато ги представяме за първи път, както и за някои коментари, добавени непосредствено към кода

`Шрифт с постоянна широчина`
: Показва съдържанието на файловете, а също изхода от командите, като същевременно обозначава модулите, методите, конструкциите и командите

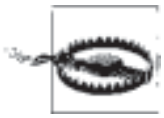
`Получерен шрифт с постоянна широчина`
: В частите от текста, в които се намира кодът, показва командите или текста, който трябва да въведете

`Курсивен шрифт с постоянна широчина`
: Показва заменяемите конструкции в съдържащите код части от текста

<Шрифт с постоянна широчина>

Представя синтактични единици, които трябва да заместите с истински код

Обозначава съвет, предложение или обща забележка, свързани с текст, намиращ се наблизо.



Обозначава предупреждение или възможна уловка, свързани с текст, намиращ се наблизо.

В използваните примери знакът %, намиращ се в началото на системния команден ред, обозначава системния промпт, независимо под коя операционна система работите (например C:\Python22> в прозорец на DOS). Недейте да пишете знака %! По същия начин при описание на поведението на интерпретатора не въвеждайте знаковете >>> и ..., показани в началото на някои от редовете – това са промптовете, визуализирани от Python. Напишете само текста, който идва след тези промптове. За ваше улеснение въведеният от потребителите код е показан с удебелен шрифт. Не е необходимо също да пишете текста, който започва със знака #; както ще прочетете по-нататък в книгата, това са коментари, а не изпълним код.

Относно програмите в книгата

Книгата, която държите в ръцете си, както и всички примери с програми, съдържащи се в нея, са базирани на версия 2.2 на Python и отразяват също предстоящата версия 2.3. Но тъй като ще се придържаме по-основно към сърцевината на езика, до голяма степен можете да сте сигурни, че казаното в текста ще важи и за следващите версии на Python. Текстът в голяма част от книгата се отнася също и към по-ранни версии на Python; съвсем естествено, ако опитате да използвате допълнителни възможности, добавени след излизането на вашата версия, няма вероятност да успеете. Като практическо правило, последната версия на Python е също и най-добрият Python. Тъй като основната насока на тази книга е сърцевината на езика, голяма част от казаното се отнася също и за Jython, базираната на Java версия на Python, както и за други версии на Python, описани в глава 2.

Можете да изтеглите сорс кода на примерите в книгата, както и кода с отговорите на упражненията от нейния уебсайт на адрес *http://www.oreilly.com/catalog/lpython2/*. Но как се стартират и изпълняват примерите? Ще навлезем в повече подробности относно стартирането на кода в глава 3, затова ви съветваме да продължите с четенето, за да научите повече.

Използване на примерите, съдържащи код

Целта на тази книга е да ви помогне в работата ви. Като правило, можете да използвате кода от книгата във вашите програми и документация. Няма нужда да се свързвате с нас, за да искате нашето съгласие, освен ако не възнамерявате да препечатвате голяма част от

кода. Така например за написването на програма, използваща няколко отрязъка от кода в книгата, не е необходимо разрешение от наша страна. От друга страна, за продажбата или разпространението на CD-ROM с примери от книгите на O'Reilly *е необходимо* разрешение. Ако отговаряте на въпрос, цитирайки кода от примерите, не се нуждаете от разрешение от авторите.

Ако искате да включите значителна част от примерния код, съдържащ се в книгата, в документацията към вашия продукт, *трябва да поискате* нашето разрешение.

Бихме оценили указване на книгата, но не го изискваме задължително. Указването обикновено включва заглавие, име на автор, издател, а също и ISBN. Например: *„ActionScript: Пълно ръководство, Второ издание, от Colin Moock. Copyright 2001 O'Reilly & Associates, Inc., 0-596-00369-X"*.

Ако считате, че начинът, по който искате да използвате примерите в книгата, не се съдържа в казаното по-горе, свържете се с нас на електронен *адрес: permissions@oreilly.com*.

Как да се свържете с нас

Изпращайте коментарите и въпросите си относно книгата на нейния издател

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (от САЩ или Канада)
(707) 829-0515 (международни или местни обаждания)
(707) 829-0104 (факс)

Книгата има също и Интернет страница, където описваме грешките в текста, примерите, а също така и допълнителна информация. Можете да посетите тази страница на адрес:

http://www.oreilly.com/catalog/lpython2

За коментари или технически въпроси, свързани с книгата, изпратете имейл на адрес:

bookquestions@oreilly.com

За повече информация относно книгата, предлаганите конференции, центровете с ресурси или O'Reilly Network, вижте нашия уебсайт:

http://www.oreilly.com

Mark и David с радост ще отговорят на въпросите на читателите, свързани с книгата, въпреки че е най-вероятно да получите от Mark отговор на въпросите, отнасящи се до „Основната част на езика" и от David – за „Външните слоеве", тъй като те съответно са автори на двата раздела. Можете да откриете имейл адресите и на двамата автори на уеб сайта на книгата.

(В текста обикновено използваме местоимението „ние", за да укажем и двама ни, но понякога, за да разкажем някой анекдот или интересна случка, използваме личните си

имена – Mark за частите от раздела „Основна част на езика” и David за раздела „Външните слоеве”, уточняващи авторство на тези части от книгата. Въпреки че в основата си книгата е колективно творение, всеки от авторите на моменти се откроява в колектива.)

За коментари и въпроси относно българското издание на книгата можете да се обърнете към издателска къща ЗеСТ Прес:

ИК “ЗеСТ Прес”
ул. Д. Манов 75
София, 1408

Информация относно други книги, издадени от ЗеСТ Прес, можете да намерите на следния адрес:

http://www.zest-press.com/,

а за въпроси и предложения можете да използвате електронната поща:

info@zest-press.com.

Благодарности

Бихме желали да изразим признанието си на всички хора, които взеха участие при написването на книгата. На първо място искаме да благодарим на редакторите, които работиха при този проект: Laura Lewin, Paula Ferguson и finally Linda Mui. Бихме искали също да благодарим на издателство O’Reilly, затова че подкрепи още един проект за книга за Python. Радваме се, че сме част от вече цялостната и все още разрастваща се тематична серия на O’Reilly, посветена на Python.

Благодарим на всички, които участваха в първоначалния преглед на книгата – Guido van Rossum, Alex Martelli, Anna Ravenscroft, Sue Giller и Paul Prescod.

А за създаването на толкова приятен за използване и полезен език дължим особено признание на Guido, на цялата Python общност; както и на доста други системи с отворен сорс, Python е плод на героичните усилия на много хора.

Искаме да изкажем специални благодарности на първоначалния редактор на книгата, покойния Frank Willison. Frank изигра голяма роля по отношение както на самия Python, така и за професионалното развитие на двамата автори. Няма да бъде пресилено, ако кажем, че на Frank се дължи голяма част от успеха на Python в първите години след неговото създаване. Всъщност дори самата книга, която държите в ръцете си, беше негова идея. Като знак за признателност за неговия широк поглед и приятелство, му посвещаваме това второ осъвременено издателство. (Продължавай напред, Frank.)

Mark казва също:

Когато за пръв път се запознах с Python през 1992 година, нямах представа какво въздействие щеше да окаже това върху следващото десетилетие от моя живот. След като през 1995 година написах първото издание на Програмиране с Python, започнах да пътувам из

страната и по света, като преподавах Python както на начинаещи потребители, така и на опитни програмисти. След приключването на първото издание на настоящата книга през 1999 станах независим писател и преподавател на Python в голяма степен благодарение на прогресивно нарастващата популярност на езика.

Към момента, в който пиша тези думи в началото на 2003 година, съм водил близо 90 работни групи за изучаване на Python в САЩ, Европа, Канада и Мексико, като през това време имах над хиляда студенти. Освен че събрах бонуси като редовен клиент на авиолиниите, тези групи ми помогнаха да допреработя и доизясня материала от книгата, свързан с основната част на езика. Като цяло, тези части от книгата са заимствани от материалите за водените от мен курсове.

Бих искал да благодаря на студентите, които участваха в групите, водени от мен през последните седем години. Предоставената от вас обратна връзка, в съчетание с настъпилите по-нови промени в Python изиграха огромна роля при оформянето на моя принос към книгата. Нищо не е по-поучително от това да видиш хиляда студенти, повтарящи едни и същи грешки! Разделът от второто издание, посветен на основната част на езика, дължи промените си главно на класовете, които водих след 1999 година; бих искал да изтъкна компаниите Hewlett-Packard, Intel и Seagate, в които проведох множество курсове през този период; бих желал да благодаря и на частните лица, които организираха учебните групи в Дъблин, Мексико сити, Барселона и Пуерто Рико; човек трудно би могъл да си представи по-големи веселяци.

Искам да изкажа признателността си на издателство O'Reilly, затова че ми даде възможност да работя върху вече шест проекта за книги; за мен беше истинско удоволствие (и само донякъде напомняше филма *Денят на мармотите*). Бих желал да благодаря и на моя съавтор, David Ascher, за неговото търпение и работата по този проект. Освен настоящата книга и целодневната му работа, в която разработва инструменти за ActiveState, David отделя от времето си и за да организира конференции, да редактира други книги, както и за много други неща.

И накрая, няколко лични благодарности. Искам да благодаря на всички хора, с които работих в различни компании в ранните години от кариерата си. На обществената библиотека Boulder, в която намерих „убежище", за да напиша някои части от книгата. На покойния Carl Sagan, затова че ме вдъхновяваше през по-младите ми години. На Jimmy Buffet за погледа над нещата, който ми даде, при навлизането ми в зрялата възраст. На една жена от Ню Мексико, с която заедно пътувахме до Оклахома, затова че ми напомни за значението на мечтите. На Denver Broncos, затова че спечелиха големия шампионат (два пъти). На Sharp и Sony, понеже произвеждат толкова добра техника. И най-вече на децата ми, Michael, Samantha и Roxanne, благодарение на които се чувствам наистина богат.

Лонгмънт и Боулдър, Колорадо
юли 2003 година

David казва също:

В добавка към по-горните благодарности бих желал да изкажа признателността си на следните хора:

На първо място искам да благодаря на Mark Lutz, затова че ме покани да работя заедно с него върху тази книга, а също и затова че ме подкрепяше като преподавател по Python. Бих желал да благодаря и на много хора, които помогнаха при първоначалните ми усилия да разбера езика Python – особено Guido, Tim Peters, Don Beaudry и Andrew Mullhaupt. Учудващо е как малка подкрепа, оказана в подходящото време, може да даде дълготрайни резултати.

Първоначално преподавах Python и Jython, което Mark продължава да прави и до днес. Студентите във водените от мен групи ми помогнаха да разбера кои части от езика са по-трудни за научаване, а също ми напомниха и за аспектите на езика, които правят използването му толкова приятно, бих желал да им благодаря за получената обратна връзка и подкрепа. Искам да изкажа признателността си и на хората, които ми позволиха да водя тези курсове: Jim Anderson (Университета Brown), Cliff Dutton (който тогава работеше в Distributed Data Systems), Geoffrey Philbrick (по онова време в Hibbitt, Karlsson & Sorensen, Inc.), Paul Dubois (Lawrence Livermore National Labs) и Ken Swisz (от KLA-Tencor). Макар вече да не преподавам Python през цялото време, мога да разчитам на придобития опит, когато обучавам начинаещи потребители.

Благодаря на научните си ръководители Jim Anderson, Leslie Welch и Norberto Grzywacz, които подкрепяха усилията ми, свързани с Python, и по-конкретно с настоящата книга, без да са напълно сигурни защо я пиша, но въпреки това ми помагаха. Всеки един от научените от тях уроци все още важи и днес.

Първите жертви на усилията ми по „проповядването" на Python заслужават особено признание, затова че изтърпяха прекомерния ентусиазъм (някои дори биха го нарекли фанатизъм) в началните години от преподавателската ми кариера: Thanassi Protopapas, Gary Strangman и Steven Finney. Благодаря на Thanassi и за полезната обратна връзка, която получих за ранните ръкописи на книгата. През последните три години няколко „активатори" (по-известни на обществото като „служители на компанията ActiveState") се оказаха невероятни колеги и приятели– бих искал да изтъкна Mark Hammond, Trent Mick, Shane Caraveo и Paul Prescod.

Научил съм много от всеки от тях – относно Python, както и за други неща. Компанията ActiveState ми предложи отлична среда за работа – в която да развивам кариерата си, да научавам нови неща от невероятни хора и да продължавам да програмирам с Python.

Благодаря на семейството си: на моите родители JacSue and Philippe, затова че винаги ме подкрепяха да следвам желанията си; на брат ми Ivan, защото ми припомни някои от първите ми срещи с текстове за програмиране (след продължителни часове напразни усилия, както и осъзнаването на факта, че програма, описана в списание Byte, е пълна с грешки, които накараха едно тринадесетгодишно момче да се разплаче от яд); благодаря на моята съпруга Emily за постоянната подкрепа, а също и за вярата, че писането на книга е нещо, с което ще се справя; на децата ми – Hugo and Sylvia, затова че споделяха ком-

пютрите си с мен – те работят с компютър с такава лекота, че нямам търпение да видя какво ще създаде тяхното поколение.

И накрая, с мисъл за второто издание на книгата, бих искал да изкажа признателност на всеки, който е допринесъл за развитието на общността на Python. Разликата между Python днес и преди пет години е огромна – самият език не се е променил особено, но светът, в който той се развива, е станал по-богат и разнообразен. Този свят прелива от ентусиазъм, код и идеи (както от брилянтни учени, така и от големи чудаци), запазвайки взаимното уважение и добро настроение. Нека в този дух продължим напред.

Ванкувър, Британска Колумбия, Канада
ноември 2003 година

Благодарности от ЗеСТ Прес

Издателска къща ЗеСТ Прес благодари на екипа, участвал в създаването и подготвянето на превода на тази книга. За добре направения превод, отговорното отношение и предаването му в срок благодарим на Валентин Димитров; на Христо Николов – за научната редакция; на Валя Тончева – за всеотдайността, с която продължава да отстоява своята позиция да пишем и говорим на български език, въпреки мощното нашествие на компютърната терминология в нашето всекидневие; на Антоанета Башева – за прецизното й изпълнение при странирането на книгата. Благодарим и на всички, участвали в създаването на книгата и нейното оформление.

ГЛАВА 9
Проверки if

Тази глава представя конструкцията `if` на Python – основната конструкция, която се използва за избор между алтернативни действия, основаващ се на резултати от проверки. Тъй като това е първият път, когато се срещаме със *съставни конструкции* – конструкции, в които са вградени други конструкции – тук ще разгледаме и основните концепции, върху които се основава моделът на синтаксиса на конструкциите на Python. И тъй като конструкцията `if` въвежда понятието проверка, ще използваме тази глава и за да изследваме идеята за проверки за истина, както и булевите изрази изобщо.

Конструкции if

С по-прости думи, конструкцията `if` на Python избира действията, които да изпълни. Тя е първичният инструмент за избор в Python и представя голяма част от *логиката*, на която всяка програма на Python се подчинява. Това е и първата ни съставна конструкция; както всички съставни конструкции на Python, `if` може да съдържа други конструкции, включително други `if`. Всъщност Python позволява да съчетавате конструкциите в неговите програми както последователно (така че да се изпълнят една след друга), така и произволно вложени (така че да се изпълняват само при определени условия).

Основен формат

Конструкцията `if` на Python се среща в повечето процедурни езици. Тя приема формата на проверка `if`, следвана от една или повече избираеми проверки `elif` (което означава „освен ако”) и завършва с избираем блок `else`. Всяка проверка, както и `else`, е свързана с блок от вложени конструкции, въведен с отместване под заглавен ред. Когато стартирате конструкцията, Python изпълнява свързания с първата проверка блок с код, която изчислява като истина, или блока `else`, ако всички проверки дадат лъжа. Основната форма на `if` изглежда по следния начин:

```
if <test1>:                    # Проверка if
    <statements1>              # Свързан блок
elif <test2>:                  # Избираеми elif
    <statements2>
else:                          # Избираема else
    <statements3>
```

Примери

Нека видим няколко прости примера на конструкцията `if` в действие. Освен първоначалната проверка `if` и свързаните с нея конструкции, всички други части са избираеми; в най-простия случай останалите части се пропускат:

```
>>> if 1:
...     print 'true'
...
true
```

Забележете, че промптът се променя на „`...`” за следващите редове в базисния интерфейс, който използваме тук (в IDLE просто ще „слезете” на отместен ред – натиснете клавиша Backspace, за да преминете на нормален ред); след като завърши един празен ред, изпълнява цялата конструкция. Не забравяйте, че `1` е булевият символ за истина, така че проверката на тази конструкция винаги дава истина; за да покриете и резултат лъжа, кодирайте и `else`:

```
>>> if not 1:
...     print 'true'
... else:
...     print 'false'
...
false
```

А сега вижте един пример с най-сложния вид, който може да приеме конструкцията `if` – в него присъстват всичките й избираеми части:

```
>>> x = 'killer rabbit'
>>> if x == 'roger':
...     print "how's jessica?"
... elif x == 'bugs':
...     print "what's up doc?"
... else:
...     print 'Run away! Run away!'
...
Run away! Run away!
```

Тази многоредова конструкция се простира от реда с `if` до блока `else`. Когато я стартирате, Python изпълнява конструкциите, вложени под първата проверка, ако тя е истина, или частта `else`, ако всички проверки са лъжа (а в този пример те са лъжа). На практика както частта `elif`, така и `else` могат да се пропуснат, а във всяка секция може да се вложи повече от една конструкция. Нещо повече, думите `if`, `elif` и `else` са свързани и с факта, че се подреждат вертикално една под друга, с едно и също отместване.

Множествен преход

Ако сте използвали езици като C или Pascal, ще ви бъде интересно да научите, че в Python няма конструкция „ключ” или case, която да избира действие въз основа на стойност на променлива. Вместо това *множественият преход* се кодира или като последователност от проверки `if/elif`, както направихме в последния пример, или чрез индексиране на речници и списъци за търсене.

Тъй като речниците и списъците могат да се изграждат в процеса на работата с тях, понякога те се оказват по-гъвкави от изричната логика на `if`:

```
>>> choice = 'ham'
>>> print {'spam':  1.25,    # "Ключ", базиран върху речник
...        'ham':   1.99,    # Използва has_key() или get()
                             # по подразбиране.
...        'eggs':  0.99,
...        'bacon': 1.10}[choice]
1.99
```

Въпреки че, когато за пръв път се натъкнете на него е необходимо малко време, за да разберете казаното, този речник е множествен преход – индексирането на ключа `choice` довежда до разклонение към няколко стойности – също като „ключ" в C. Почти равносилната и „по-многословна" конструкция `if` на Python би изглеждала по следния начин:

```
>>> if choice == 'spam':
...print 1.25
... elif choice == 'ham':
...print 1.99
... elif choice == 'eggs':
...print 0.99
... elif choice == 'bacon':
...print 1.10
... else:
...print 'Bad choice'
...
1.99
```

Обърнете внимание, че тук клаузата `else` към `if` поема случая *по подразбиране*, ако никой от ключовете не съвпадне. Както видяхме в глава 6, стойностите по подразбиране на даден речник могат да се кодират с проверки `has_key`, с извиквания на метода `get` или чрез прихващане на изключенията. Тук всеки от посочените инструменти може да се използва, за да се кодира действие по подразбиране в множествен преход, базиран върху речник. Ето случай с `get` в действие със стойности по подразбиране:

```
>>> branch = {'spam': 1.25,
...           'ham':  1.99,
...           'eggs': 0.99}

>>> print branch.get('spam', 'Bad choice')
1.25
>>> print branch.get('bacon', 'Bad choice')
Bad choice
```

Речниците са удобни за свързването на стойности с ключове, но дали е така и при по-сложните действия, които можете да кодирате с конструкции `if`? В част IV ще научите, че речниците могат да съдържат и *функции*, чрез които да представят по-сложни разклонени операции, както и да изграждат основни таблици за преход. Тези функции приемат формата на стойности на речника и често се кодират като `lambda` изрази, които се извикват с добавяне на скоби, за да се стартира действието им.

Правила на синтаксиса на Python

Като цяло Python има прост, основаващ се на конструкции синтаксис. Но все пак се налага да знаете някои характеристики:

Конструкциите се изпълняват една след друга, докато не укажете нещо различно. Python обикновено изпълнява конструкциите – от първата до последната – от файл или от вложен блок, но конструкции от рода на `if` (и циклите, както ще видите) карат интерпретатора да „подскача" напред-назад из вашия код. Тъй като пътят, който Python изминава през всяка програма, се нарича управление на изпълнението, променящи го фактори като `if` се наричат конструкции за контрол на изпълнението.

Границите на блоковете и на конструкциите се откриват автоматично. В блоковете, съдържащи код, няма скоби или разделители за „начало" и за „край"; вместо това Python използва отместването на конструкциите под заглавен ред, за да ги групира във вложен блок. По същата логика конструкциите на Python обикновено не завършват с точка и запетая; вместо това краят на реда обикновено маркира края на конструкциите, кодирани на него.

Съставните конструкции се състоят от заглавен ред, ":" и отместени навътре конструкциии. Всички съставни конструкции в Python спазват един и същ модел: заглавен ред, който завършва с двоеточие, следван от една или повече вложени конструкции, написани с отместване под заглавния ред. Отместените конструкции се наричат *блок* (в някои случаи – пакет). При конструкцията `if` клаузите `elif` и `else` са част от `if`, но представляват заглавни редове със собствени вложени блокове.

Празните редове, интервалите и коментарите обикновено се игнорират от Python. Празните редове във файловете (но не и в интерактивния промпт) се игнорират. Интервалите в конструкциите и изразите почти винаги се пренебрегват (освен при низовите литерали и отместването). Коментарите винаги се игнорират: те започват със знак # (но не вътре в низов литерал) и се простират до края на текущия ред.

Документационните знакови низове се пренебрегват, но се съхраняват и визуализират с инструменти. Python поддържа допълнителна форма за коментари, наречена документационни низове (или накратко *docstrings*), които за разлика от коментарите с # се запазват. Документационните низове са просто знакови низове, които се появяват в горната част на програмните файлове и на някои конструкции, като автоматично се свързват с обекти. Съдържанието им се пренебрегва от Python, но те автоматично се прикачат към обекти в процеса на изпълнение и могат да се визуализират с документационни инструменти. Документационните низове са част от по-обширната документация на Python и се разглеждат в края на част III.

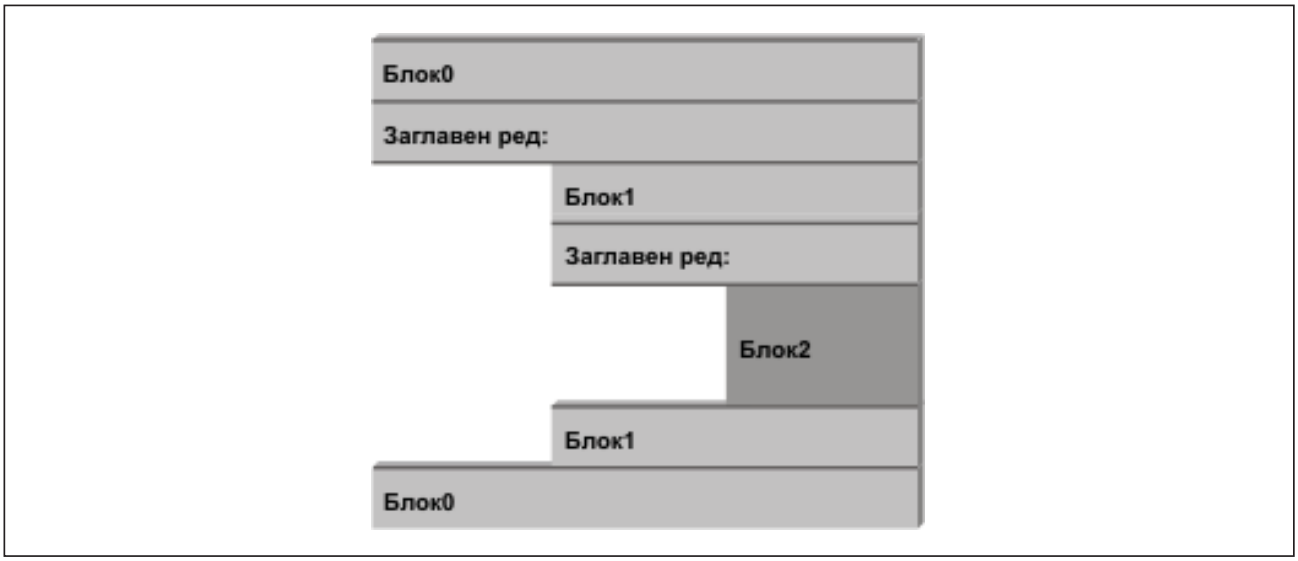
Както вече видяхте, в Python няма деклариране на типа на променливите; този факт сам по себе си значително опростява синтаксиса на езика, в сравнение с този, с който може би сте свикнали. Но повечето от новите потребители възприемат липсата на скоби и на точки и запетаи за обозначаване на блоковете и конструкциите като най-необичайната синтактична черта на Python, затова нека разгледаме в повече подробности какво представлява тя.

Разделители за блокове

Python открива автоматично границите на блоковете по *отместването* на редовете – празното пространство от лявата страна на вашия код. Всички конструкции, които са отместени на еднакво разстояние надясно, принадлежат към един и същ блок с код. С други думи, конструкциите, намиращи се вътре в самия блок, са вертикално подравнени. Блокът завършва с по-малко отместен ред или с края на файл, а по-дълбоко вложените блокове просто са отместени още по-надясно от конструкциите на съдържащия ги блок.

Така например фигура 9.1 онагледява блоковата структура на следващия код:

```
x = 1
if x:
    y = 2
    if y:
        print 'block2'
    print 'block1'
print 'block0'
```



Фигура 9.1. Вложени блокове с код

Този код съдържа три блока: първият (най-високото ниво на файла) не е отместен изобщо; вторият блок (във външната конструкция `if`) е отместен с четири интервала; а третият блок (конструкцията `print` под вложената `if`) е отместен с осем интервала.

Като цяло кодът от най-високо ниво (невложен) трябва да започва от колона 1. Вложените блокове могат да започват от която и да е колона; отместването може да се състои от произволен брой интервали и табулации, стига то да е едно и също за всички конструкции в даден блок. Тоест Python не се интересува как отмествате кода си; той само изисква това да става последователно. Технически погледнато, табулациите се равняват на достатъчен брой интервали за преместването на номера на текущата колона с кратно на 8 число, но обикновено не е разумно да смесвате табулации и интервали в един и същ блок – използвайте или едното, или другото.

Отместването от лявата страна на вашия код е единственото основно място в Python, където празното пространство има значение; в повечето други случаи е все едно дали празното пространство се кодира, или не. Въпреки това отместването действително е част от синтаксиса на Python, а не просто стилистично предложение: всички конструкции вътре в един и същ блок трябва да бъдат отместени еднакво, в противен случай Python съобщава за синтактична грешка. Това е направено умишлено – тъй като няма нужда изрично да обозначавате началото и края на вложения блок с код, така се премахва част от синтактичния безпорядък, който се среща в други езици.

Синтактичният модел налага също последователност при отместването, важно условие за четливостта в структурираните езици за програмиране като Python. Понякога за синтаксиса на Python казват: „Това, което виждате, е и това, което получавате.” – отместването на кода недвусмислено указва на читателите с какво е свързано. Последователният „изглед” на Python улеснява поддръжката на кода.

Последователно отместеният код винаги задоволява правилата на Python. Освен това повечето текстови редактори (включително IDLE) улесняват спазването на модела на отместване на Python, като автоматично отместват кода, докато го пишете.

Разделители за конструкции

Конструкциите обикновено завършват накрая на реда, на който се появяват. Това важи за голяма част от конструкциите на Python, които ще кодирате. Когато конструкциите са твърде дълги, за да се съберат на един ред, могат да се използват няколко специални правила, които им позволяват да се разпростират на няколко допълнителни реда:

Конструкциите могат да обхващат няколко реда, ако продължавате отворена синтактична двойка. При конструкциите, които са твърде дълги, за да се поберат на един ред, Python ви позволява да продължите да пишете конструкцията на следващия ред, ако кодирате нещо, което е затворено с двойки от `()`, `{}` или `[]` скоби. Така например изразите в обикновени скоби и литералите за речници и списъци могат да се простират на произволен брой редове; вашата конструкция не завършва, докато не напишете затварящата част на двойката (`)`, `}` или `]`). Допълнителните редове могат да започват с каквото и да е ниво на отместване.

Конструкциите могат да обхващат няколко реда, ако завършват с обратно наклонена черта. Това е донякъде остаряла възможност, но ако е необходимо дадена конструкция да бъде продължена на няколко реда, можете да добавите обратно наклонена черта (\) в края на предишния ред, за да укажете, че ще продължите и на следващия. Но тъй като можете да продължите също и като оградите със скоби дълги конструкции, обратно наклонените черти почти винаги са излишни.

Други правила. Много дългите низови литерали могат да се простират на произволен брой редове. Всъщност зададените с тройни кавички низови блокове, които срещнахме в глава 5, са умишлено създадени с тази възможност. Можете да завършвате програмите и с точка и запетая, макар че този начин се среща по-рядко – понякога се използва, за да се поберат повече от една прости конструкции на един ред. И накрая, коментарите и празните редове могат да се появяват на което и да е място.

Няколко по-особени случая

Ето как изглежда продължаващият ред, когато използваме правилото за отворените синтактични двойки; можем да разпростираме граничните конструкции на произволен брой редове:

```
L = ["Good",
     "Bad",
     "Ugly"]                    # Отворените двойки позволяват
                                # разпростиране на няколко реда.
```

Това се отнася и за всичко, което се намира в обикновени скоби: изрази, аргументи на функции, заглавни части на функции (вижте глава 12) и т.н. Ако ви харесва да използвате обратно наклонена черта, за да продължите на следващия ред, можете да го направите, но обикновено не е задължително:

```
if a == b and c == d and   \
   d == e and f == g:
   print 'olde'              # Обратно наклонените черти позволяват
                            # продължение на следващ ред.
```

Тъй като всеки израз може да бъде затворен в скоби, обикновено можете да ограждате нещо в скоби всеки път, когато искате да продължите на няколко реда:

```
if (a == b and c == d and
    d == e and e == f):
    print 'new'               # Но в повечето случаи скобите
                            # позволяват същото.
```

Като особен случай, Python ви позволява да напишете на един и същ ред повече от една конструкции, които не са съставни (тоест конструкции без вложени в тях конструкции), като ги разделяте с точки и запетаи. Някои програмисти използват тази форма, за да спестят „площ" от програмния файл, но кодът става далеч по-четлив, ако пишете по една конструкция на ред в по-голямата част от програмите си:

```
x = 1; y = 2; print x       # Повече от една проста конструкция.
```

И накрая, Python позволява да изместите тялото на съставна конструкция на заглавния ред, при положение че то представлява проста конструкция (а не друга съставна конструкция). Това се използва най-често при прости конструкции `if` само с по една проверка и действие:

```
if 1: print 'hello'         # Проста конструкция на заглавния ред.
```

Можете да съчетаете няколко от гореизброените особени случая, за да пишете по-нечетлив код, но не ви съветваме да го правите; препоръчваме ви в практиката да се стремите да пишете всяка конструкция на отделен ред, като използвате отместване при всички блокове, освен при най-простите от тях. Шест месеца по-късно ще се радвате, че сте избрали този начин на действие.

Проверки за истинност

Вече въведохме понятията сравнение, равенство и стойност на истинност в глава 7. Тъй като конструкцията `if` е първата, която действително използва резултати от проверки, тук ще разгледаме по-обстойно някои от тези понятия. По-специално булевите оператори на Python леко се различават от своите „двойници" в езици като C. В Python:

- Истина означава всяко число, различно от нула, и всеки обект, който не е празен.
- Неистина е обратното на истина: нулево число, празен обект или `None`.
- Сравненията и проверките за равенство се прилагат повтаряемо към структурите от данни.
- Сравненията и проверките за равенство връщат стойности 1 и 0 (истина и неистина).
- Булевите оператори `and` и `or` връщат обект-операнд истина или неистина.

Накратко булевите оператори се използват за комбинирането на резултатите от други проверки. В Python съществуват три булеви израза с оператори:

`X and Y`
: Истина е, ако както `X`, така и `Y` са истина.

`X or Y`
: Истина е, ако или `X`, или `Y` е истина.

`not X`
: Истина е, ако `X` е неистина (изразът връща 1 или 0).

Тук `X` и `Y` могат да бъдат произволна стойност на истинност или израз, който връща стойност на истинност (например проверка за равенство, сравнение за интервал и т.н.). В Python булевите оператори се изписват като думи (за разлика от `&&`, `||` и `!` в C). В Python булевите оператори `and` и `or` връщат *обект* истина или неистина, а не цели числа 1 или 0. Нека разгледаме няколко примера, за да видим как става това:

```
>>> 2 < 3, 3 < 2                # По-малко от: връща 1 или 0
(1, 0)
```

Сравненията за големина като показаното връщат цяло число 1 или 0 като стойност на истинност. Но операторите `and` и `or` винаги връщат обект като резултат. При проверките or Python изчислява обектите-операнди отляво надясно и връща първия, който е истина. Нещо повече, Python спира при първия намерен операнд, който е истина; това обикновено се нарича *съкратено оценяване*, тъй като определянето на резултат съкращава (приключва) останалата част от израза:

```
>>> 2 or 3, 3 or 2              # Връща левия операнд, ако е истина.
(2, 3)                          # В противен случай връща десния операнд
                                #  (истина или неистина).
```

```
>>> [] or 3
3
>>> [] or {}
{}
```

В първия ред от горния пример и двата операнда са истина (2, 3), затова Python спира и връща операнда, който се намира отляво. При другите две проверки левият операнд е неистина, затова Python просто оценява и връща обекта, намиращ се отдясно (който ако бъде проверен, ще има стойност истина или неистина). Освен това операциите `and` спират веднага, щом резултатът им стане известен; в този случай Python оценява операндите отляво надясно и спира при първия обект, който е неистина:

```
>>> 2 and 3, 3 and 2          # Връща левия операнд, ако е неистина.
(3, 2)                        # В противен случай връща десния операнд
                              # (истина или неистина).
>>> [] and {}
[]
>>> 3 and []
[]
```

В първия ред на този пример и двата операнда са истина, затова Python изчислява и двете страни и връща обекта, който се намира отдясно. При втората проверка левият операнд е неистина (`[]`), затова Python спира и го връща като резултат от проверката. При последната проверка лявата страна е истина (`3`), затова Python оценява и връща обекта, намиращ се отдясно (който в този случай е неистина `[]`).

Окончателният резултат от всичко това е същият като в C и в повечето от другите езици – получавате резултат, който логически представлява истина или неистина, ако се провери в `if` или `while` конструкция. Въпреки това в Python булевите изрази връщат или левия, или десния обект, а не маркер цяло число.

Една последна забележка: Както споменахме в глава 7, Python 2.3 съдържа нов булев тип, наречен `bool`, който вътрешно е подклас на типа за цели числа `int`, със стойности `True` и `False`. Последните две стойности всъщност са персонализирани версии на целите числа `1` и `0`, които дават като резултат думите `True` и `False`, когато се визуализират или като се преобразуват по друг начин в знакови низове. Единственият път, когато ще забележите тази промяна, е като видите булевите изходи да се изписват във вид `True` и `False`. Повече подробности за разделянето на типовете на подкласове потърсете в глава 23.

Защо Булевите изрази са важни

Често срещан начин за използването на специалното поведение на булевите оператори на Python е да се избира между няколко обекта с `or`. Конструкцията:

```
X = A or B or C or None
```

задава на `X` първия от обектите `A`, `B` и `C`, който не е празен (тоест истина), или `None`, ако и трите са празни. Това е доста често използван образец за кодиране в Python: за да изберете обект, който не е празен, между определен брой обекти, просто ги изредете един след друг в израз `or`.

Важно е също така да разберете и съкратеното оценяване, тъй като изразите от дясната страна на даден булев оператор могат да извикват функции, които извършват много работа, или дават странични ефекти, които няма да се задействат, ако правилото за съкратеното оценяване влезе в действие:

```
if f1() or f2(): ...
```

В този пример, ако `f1` върне стойност истина (или която не е празна), Python никога няма да изпълни `f2`. За да сте сигурни, че и двете функции ще се изпълнят, извикайте ги преди оператора `or`:

```
tmp1, tmp2 = f1(), f2()
if tmp1 or tmp2: ...
```

Друго приложение на този режим на работа ще разгледаме в глава 14: поради начина, по който се изпълняват булевите изрази, изразът `((A and B) or C)` може да се използва, за да се емулира почти напълно конструкция `if/else`. Освен това, тъй като понятията истина и лъжа са вътрешноприсъщи за всички обекти, в Python е по-лесно (и по-често срещано) обектите да се проверяват директно `(if X:)`, вместо да се сравняват с празна стойност `(if X != '':)`. Ако се приложат към знаков низ, двете проверки са равносилни.