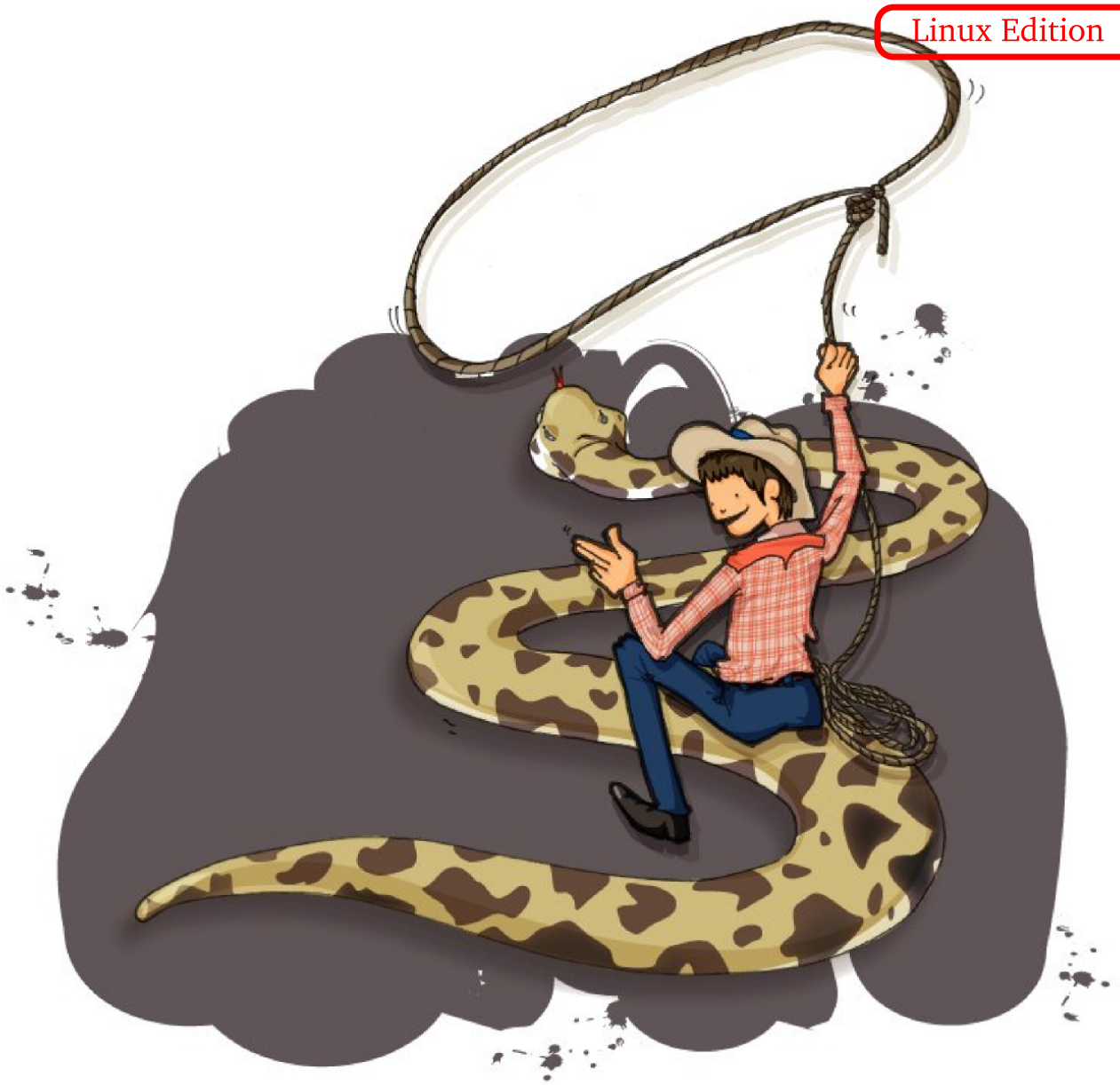
Snake Wrangling For Kids

Learning to Program with Python

Linux Edition



Written by Jason R. Briggs

Snake Wrangling for Kids, Learning to Program with Python
by Jason R. Briggs
перевод на русский:
Кочетов Е. М. <Egor.Kochetoff@gmail.com>

Version 0.7.7

Cover art and illustrations by Nuthapitol C.

This book has been completely rewritten and updated, with new chapters (including developing graphical games), and new code examples. It also includes lots of fun programming puzzles to help cement the learning. Published by No Starch Press - available here: Python for Kids. Also find more info here.

Website:
http://www.briggs.net.nz/log/writing/snake-wrangling-for-kids
Github с переводом книги

Thanks To:
Guido van Rossum (for benevolent dictatorship of the Python language), the members of the Edu-Sig mailing list (for helpful advice and commentary), author David Brin (the original instigator of this book), Michel Weinachter (for providing better quality versions of the illustrations), and various people for providing feedback and errata, including: Paulo J. S. Silva, Tom Pohl, Janet Lathan, Martin Schimmels, and Mike Cariaso (among others). Anyone left off this list, who shouldn't have been, is entirely due to premature senility on the part of the author.

License:

Оглавление

Вступление **1**

1 Не все змеи будут шипеть на тебя **3**
1.1 Пара слов про язык 5
1.2 Орден Неядовитых Удушающих Змей... 5
1.3 Первая программа на Питоне 6
1.4 Вторая программа на питоне... опять то же самое? 7

2 8 умножить на 3.57 равняется... **11**
2.1 Use of brackets and ``Order of Operations'' 13
2.2 There's nothing so fickle as a variable 14
2.3 Using Variable 16
2.4 A Piece of String? 18
2.5 Tricks with Strings 19
2.6 Not quite a shopping list 20
2.7 Tuples and Lists 23
2.8 Things to try 24

Вступление

Пара слов для родителей...

Уважаемый родитель или иной управляющий компьютером!

Чтобы ваш ребёнок смог начать знакомиться с программированием, вам нужно установить Python на компьютер. Эта книга была недавно обновлена до версии Python 3.0, самой новой и несовместимой с предыдущими, так что если у вас установлена более старая версия Pyhton, вам стоит скачать и более старую версию этой книги.

Установка Python — достаточно простая задача, но есть несколько тонкостей — в зависимости от используемой операционной системы. Если вы только что купили сверкающий новый компьютер и не имеете никаких идей, что с ним делать, а предыдущее предложение начало вызывать у вас нервную дрожь или холодный пот, то, пожалуй, лучше вам найти кого-то, кто сделает это за вас. Установка Python может занять от 15 минут до пары часов в зависимости от скорости интернета и компьютера.

Прежде всего, скачайте и установите последнюю версию Python 3 для вашего дистрибутива. Дистрибутивов очень много, так что инструкции для всех тут привести не получится... да и скорее всего, если вы используете Linux, то уже знаете, как это сделать. Наверное, вы даже возмущены самой идеей того чтобы рассказывать вам, как что-либо устанавливать, так что тут я остановлюсь.

После установки...

...Вам, возможно, придётся в течение первых пары глав посидеть с ребёнком рядом, но после нескольких примеров ему будет только приятнее читать книгу самому (с компьютером вместе). Нужно рассказать ребёнку, как вводить команды в консоль, как пользоваться текстовым редактором (наподобие блокнота; Microsoft Word никак не подойдёт), открывать и сохранять файлы в этом редакторе. Всё остальное расскажет эта книга.

Спасибо за уделённое время; с наилучшими пожеланиями,
КНИГА.

Глава 1

Не все змеи будут шипеть на тебя

Возможно, тебе подарили эту книгу на день рожденья. А может, на рождество. Например, так: тётя Агата (у всех есть тётя Агата, но не все об этом знают) хотела подарить носки, хотя и не парные, но оба красивые, на два размера больше — на вырост (и всё равно бы эти носки не пригодились потом). А потом вместо этого услышала про эту книгу (которую можно взять и напечатать), вспомнила твои вопросы про всякие компьютерные штуки, и твои непонятные объяснения, как пользоваться компьютером, оборвавшиеся в момент, когда она начала разговаривать с компьютерной мышью, и решила подарить эту книгу. Во всяком случае эта книга уж точно лучше пары разных носков.

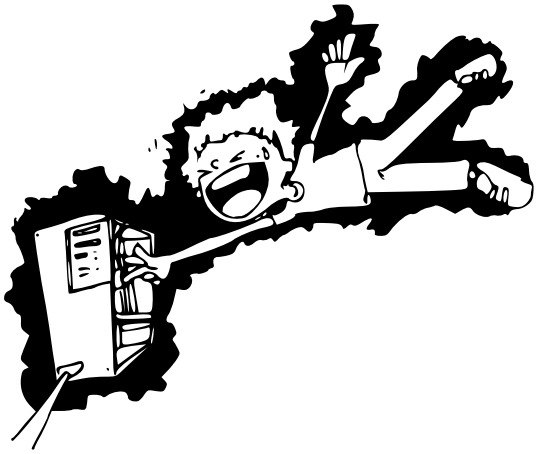
Надеюсь, я не слишком тебя разочаровываю тем, что я — возможно, напечатанная на какой-нибудь старой обёрточной бумаге (хотя если повезло, то и нет) — не слишком разговорчивая (прямо, скажем, совсем молчаливая) книга, с пугающим словом «изучение» в названии... Но представь на минутку и мои ощущения. Если бы вот ты был персонажем из какой-нибудь книги про волшебников, одна из которых наверняка есть у тебя в спальне на книжной полке, — у меня бы могли быть зубы... или даже глаза! А ещё какие-нибудь движущиеся картинки, таинственные звуки... ладно, чего я. В общем, я просто бумажная книжка, хотя могло бы быть и лучше.

Ах, много бы я дала за пару хороших острых челюстей...

Но вообще, быть конкретно такой книжкой тоже не слишком печально. Ну не могу я говорить... пальцы покусывать не могу; зато могу рассказать немного о том, что заставляет компьютеры работать. Не про разные аппаратные штуки — все эти провода, платы, чипы — они меня немного пугают. Электричеством, например, могут ударить (так что не стоит и пытаться ту-

да лезть, как по мне). Я могу рассказать о том, что удивительным образом скрыто внутри всех этих проводов, микросхем и что делает компьютер по-настоящему полезным.

Вообще, это здорово похоже на мысли, например, в твоей голове. Если бы мыслей у тебя не было — сидел бы ты, скажем, на полу в спальне и бессмысленно смотрел в пространство перед собой. Без *программ* компьютеры бы могли приносить пользу, пожалуй, разве что как стопор для двери. Да и то посредственный: вечно все бы об него спотыкались по ночам. А что может быть хуже, чем удариться ночью в темноте пальцем ноги с размаху о железный угол...



Итак, я всего лишь книга. И мне это хорошо известно.

Вообще, у тебя в семье могут быть разные устройства вроде Playstation, Xbox, Wii — игровые консоли, — а ещё DVD-проигрыватель, может, даже современный холодильник и игрушечная машинка. В них во всех есть программы, которые делают их намного полезнее, чем если бы эти штуки были без программ. В DVD-проигрывателе есть программа для чтения и воспроизведения дисков. В холодильнике — какая-то простая программа для поддержания температуры при минимальных затратах электричества. В машинке — программа для приёма команд с пульта управления и для езды в ответ на эти команды. А в настоящих машинах программы показывают маршруты в объезд пробок и сигналят водителю, когда он паркуется, чтоб он никуда не въехал (в стену или соседнюю машину).

Зная, как писать программы, ты сможешь сделать множество самых разных полезных вещей. Можно свою игру написать. Можно писать страницы в интернете, которые что-нибудь делают, а не просто показывают текст и картинки. Можно упрощать себе выполнение домашней работы.

Так вот, пора приступить к чему-то чуть более интересному, чем эти рассуждения.

1.1. Пара слов про язык

Так же как и у людей, определённо как у китов, возможно, и у дельфинов и, возможно, у родителей (тут, конечно, спорно), у компьютеров есть свой собственный язык. Вообще, как и у людей, у компьютеров много языков. Какую букву английского алфавита ни возьми, она называет какой-нибудь язык. Вот, например, буквы A, B, C, D, E — не только буквы, но и названия языков программирования (что ещё раз доказывает, что у взрослых никакого воображения и хорошо бы им давать почитать хотя бы словарь перед тем, как названия придумывать).

Есть ещё языки программирования, названные в честь людей[1], есть языки-сокращения из заглавных букв (SQL, например), есть немножко названных в честь телевизионных шоу. А, да, ещё если дописать к этим буквам всяких значков типа плюсиков, решёточек (+, #), то тоже получатся названия разных языков программирования. И ещё впридачу некоторые языки очень похожи и отличаются только в каких-то мелочах.

Что я говорила? Никакого воображения!

К счастью, многие из языков уже почти не используются или совсем исчезли; но однако же список способов «говорить» с компьютером всё ещё пугающе велик. Я буду обсуждать только один из них, потому что иначе всё так и закончится их перечислением, не успеем мы приступить к чему-то действительно интересному.

1.2. Орден Неядовитых Удушающих Змей...

... или просто питонов.

Вообще, питон — не только змея[2], но и язык программирования. Многие называют его «пайтон», как принято за рубежом, где его и придумали (и пишут его название как Python). Язык, правда, был назван не в честь змеи — это один из немногих языков программирования, названных в честь телевизионного шоу. Монти Пайтон (Monty Python) — британское телешоу, популярное с 1970х годов. Требуется достичь некоторого возраста и иметь определённый склад ума, чтобы счесть его забавным, но многим нравится. Хотя лет до 12 смотреть вообще смысла нет, будет скучно и непонятно.

Есть несколько особенностей Питона (языка программирования, а не змеи), делающих его очень полезным, чтобы учиться программировать. Для нас сейчас важнее всего то, что используя его, можно быстро сесть и начать писать

[1] например, язык Ада — в честь Ады Лавлейс. А есть ещё язык РАЯ.

[2] которая может не есть полтора года кряду!

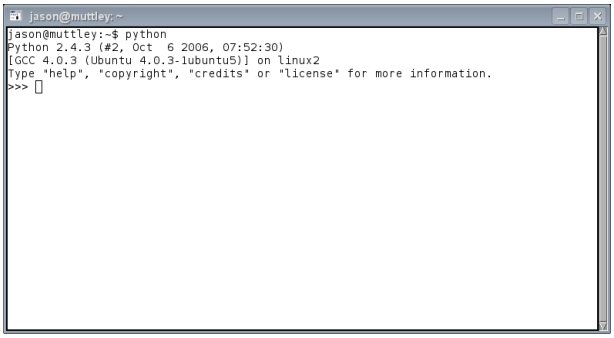


Рис. 1.1: Консоль Питона в Линуксе.

какие-то программки, пусть и очень простые, без долгих разговоров и объяснений.

Это тот момент, когда надо убедиться, что твоя мама, папа или кто там управляет компьютером прочли часть «Заметка для мам и пап». Есть хороший способ проверить это. Спроси у них, как называется программа-*эмулятор терминала* — это может быть `yakuake`, `konsole`, `rxvt`, `xterm` или ещё какая-нибудь, их очень много бывает — именно поэтому придётся спросить. Запусти эту программу, напиши *в командной строке* 'python3' (без кавычек) и нажми на клавиатуре клавишу Enter. На экране должно появиться что-то похожее на рисунок 1.1.

Если ты обнаружишь, что они не прочитали инструкции в начале книги...

... и из-за этого у тебя не получилось что-то сделать, то перелистни эту книгу на начало, подсунь им под нос введение, пока они читают утреннюю газету, и умоляюще посмотри на них. Иногда помогает говорить «пожалуйста-пожалуйста-пожалуйста» до тех пор, пока они не встанут и не сделают всё, что надо. Ну и конечно, можно попробовать сделать всё самостоятельно, это может оказаться даже проще.

1.3. Первая программа на Питоне

Так или иначе, если ты добрался досюда, у тебя уже открыта *консоль*, или командная строка Питона — это один из способов запускать команды и целые программы на Питоне. После запуска консоли или ввода любой команды ты увидишь *приглашение командной строки*, которое в Питоне выглядит вот так:

```
>>>
```

Если записать несколько команд на Питоне одну за другой, получится программа, которую можно запускать и не через консоль, но пока на минутку остановимся на простых командах, которые можно вводить прямиком в командную строку (после «приглашения»). Например, можно ввести туда следующую команду:

```
print("Всем привет!")
```

Чтобы всё получилось, нужно ввести и скобки и кавычки (вот эти: "") так, как написано выше. Тогда на экране должно появиться что-то вроде такого:

```
>>> print("Всем привет!")
Всем привет!
```

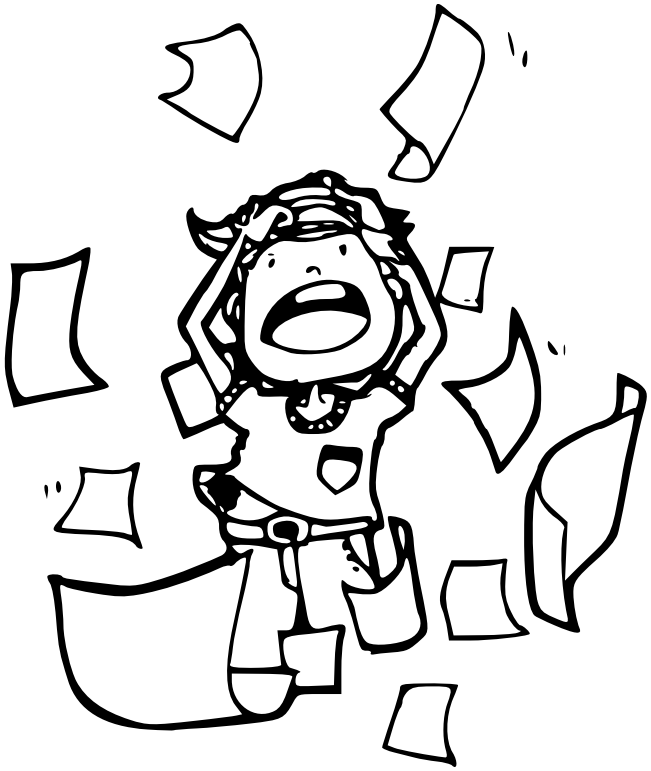
После этого приглашение командной строки появится снова, чтобы показать, что Питон готов принимать новые команды. Поздравляю! Ты только что создал и запустил свою первую программу на Питоне — пусть пока и всего из одной команды: print — функции, которая просто печатает всё, что написано в скобках. Потом мы много будет использовать эту команду.

1.4. Вторая программа на питоне... опять то же самое?

Программы на Питоне были бы не слишком полезными, если бы их приходилось каждый раз вводить заново в командную строку или если бы ты написал программу для кого-то, а ему бы пришлось её перепечатывать, чтобы запустить.

Программа для редактирования текстов (Microsoft Word, Libreoffice Writer или другая подобная), которую ты, вероятно, используешь для выполнения каких-нибудь домашних заданий, получена из исходного кода размером примерно от 10 до 100 миллионов строк. Если печатать это на бумаге с двух сторон не очень крупно, это может занять, например, 400 000 страниц. Это стопка бумаги высотой 40 метров, с десятиэтажный дом. Такое количество бумаги нести из магазина в дом, чтобы перепечатать, пришлось бы долго... очень долго...

...а если бы ещё и ветер подул в подходящий момент... за бумагой пришлось бы долго бегать. Так вот, хорошая новость: всем этим заниматься не обязательно.



Открой текстовый редактор (можешь опять спросить у родителей, как он называется: например, kate, gedit, kdevelop, но никак не Microsoft Word, он не подойдёт) и напиши туда точно ту же самую команду, что ты до этого вводил в консоль:

```
print("Всем привет!")
```

Теперь сохрани этот файл в своей домашней папке. Наверху в программе должен быть значок сохранения, а когда тебя спросят, куда сохранять, нажми на какую-нибудь кнопку типа домика. В качестве имени файла введи «hello.py». Теперь опять открой терминал и напиши:

```
python hello.py
```

В консоли должно появиться приветствие от программы, точно так же, как в прошлый раз (примерно как на рисунке 1.2).

Вот. Теперь ты видишь, что мудрые люди, создавшие Питон, спасли тебя от ввода одних и тех же программ много-много-много раз для выполнения одних и тех же действий. Как они делали в 1980х. Я серьёзно, им приходилось вводить каждый раз кучу команд для выполнения одной и той же программы. Можешь спросить у папы — вдруг у него был ZX81 в молодости — так там приходилось так делать. Теперь можно просто написать имя программы, и она целиком исполнится от начала до конца.

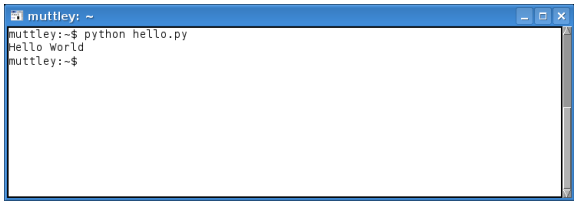


Рис. 1.2: Запуск программы на Питоне, сохранённой в текстовый файл

Конец начала

Добро пожаловать в удивительный мир программирования! Мы начали с простой программы, которая печатает «Всем привет» («Hello world») — все с этого начинают, когда учатся программировать. В следующей главе мы займёмся чуть более полезными вещами в консоли Питона, а потом изучим, как написать программу посложнее.

Глава 2

8 умножить на 3.57 равняется...

Чему равно 8 умножить на 3.57? Пришлось бы использовать калькулятор, чтобы посчитать? Ладно, этот пример можно вычислить и в уме, но не в том дело. То же самое можно сделать в консоли Питона. Запусти её опять (как описано тут: **??**), если она ещё не запущена, и введи туда такую команду: `8*3.57`. Потом нажми Enter.

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 8 * 3.57
28.56
```

Звёздочка (`*`) (обычно это shift + 8) используется для умножения вместо привычного символа × (или ·), потому что не везде их можно просто так ввести с клавиатуры, а буква X путалась бы с собственно буквой, если бы её использовали как знак умножения. Ладно, как насчёт чего-нибудь более полезного?

Suppose you do chores once a week, for which you get $5 pocket money, and you have a paper round which you do 5 times a week and get $30---how much money would you earn in a year?

If we were writing that on paper, we might write something like:

```
(5 + 30) x 52
```

Which is $5 + $30 multiplied by 52 weeks in a year. Of course, we're smart, and we know that 5 + 30 is 35, so the equation is really:

```
35 x 52
```

Which is easy enough to do in a calculator, or on paper. But we can do all of these calculations with the console as well:

Питон сломался!?!?

If you just picked up a calculator and entered 8 x 3.57 the answer showing on the display will be:

28.56

Why is Python different? Is it broken?

Actually, no. The reason can be found in the way floating point (fractional numbers with a decimal place) numbers are handled by the computer. It's a complicated and somewhat confusing problem for beginners, so it's best to just remember that when you're working with fractions (i.e. with a decimal place on a number), *sometimes* the result won't be exactly what you were expecting. This is true for multiplication, division, addition or subtraction.

```
>>> (5 + 30) * 52
1820
>>> 35 * 52
1820
```

So, what if you spend $10 each week? How much do you have left at the end of the year? We could write the equation on paper a couple of different ways, but let's just type it into the console:

```
>>> (5 + 30 - 10) * 52
1300
```

That's $5 and $30 minus $10 multiplied by 52 weeks in the year. And you'd have $1300 left at the end of the year. Okay, so that's not looking all that useful so far. We can do all of that with a calculator. But we'll come back to this later and show how to make it much more useful.

You can do multiplication and addition (obviously), and subtraction and division in the Python console, along with a bunch of other maths operations that we won't go into now. For the moment the basic maths symbols for Python (actually they're called operators) are:

+	Addition
-	Subtraction
*	Multiplication
/	Division

The reason the forward-slash (/) is used for division, is that it would be rather difficult to draw a division line (plus they didn't bother to put a division symbol ÷ on the computer keyboard) as you're supposed to use for written equations. For example if you had 100 eggs and 20 boxes, you might want to know how many eggs would go in each box, so you'd show dividing 100 into 20, by writing the following equation:

$$\frac{100}{20}$$

Or if you know about long division, you might write it out fully like this:

$$\begin{array}{r|l} 100 & 20 \\ \cline{2-2} 100 & 5 \\ \cline{1-1} 0 & \end{array}$$

Or you might even write it like this:

$$100 \div 20$$

However, in Python terms you would just type it as ``100 / 20''.
Which is much easier, I think. But then, I'm a book---what do I know?

2.1. Use of brackets and ``Order of Operations''

We use brackets in a programming language to control what is called ``Order of Operations''. An operation is the use of an operator (one of those symbols in the table above). There are more operators than those basic symbols, but for that simple list (addition, subtraction, multiplication and division), it's enough to know that multiplication and division both have a higher order than addition and subtraction. Which means you do the multiplication or division part of an equation before you do the addition or subtraction part. In the following equation, all the operators are addition (+), the numbers are added in order:

```
>>> print(5 + 30 + 20)
55
```

Similarly, in this equation, there are only addition and subtraction operators, so again Python considers each number in the order it appears:

```
>>> print(5 + 30 - 20)
15
```

But in the following equation, there is a multiplication operator, so the numbers 30 and 20 are considered first. The equation is another way of saying, ``multiply 30 by 20, then add 5 to the result'' (multiplication first, because it has a higher order than addition):

```
>>> print(5 + 30 * 20)
605
```

So what happens when we add brackets? The following equation shows the result:

```
>>> print((5 + 30) * 20)
700
```

Why is the number different? Because brackets control the order of operations. With brackets, Python knows to calculate using the operators in the brackets first, then do the operators outside. So that equation is another way of saying, ``add 5 and 30, then multiply the result by 20''. The use of brackets can become more complicated. There can be brackets inside brackets:

```
>>> print(((5 + 30) * 20) / 10)
70
```

In this case, Python evaluates the **inner** most brackets first, then the outer brackets, and then the other operator. So this equation is a way of saying, ``add 5 and 30, then multiply the result by 20, finally divide that result by 10''. The result without brackets is again slightly different:

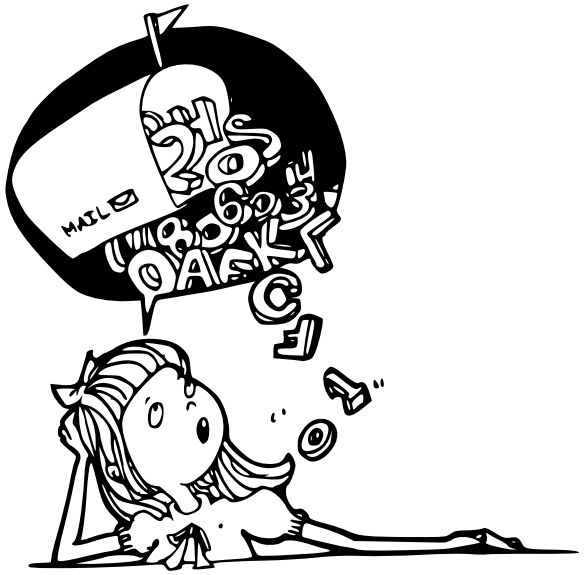
```
>>> 5 + 30 * 20 / 10
65
```

In this case 30 is multiplied by 20 first, then the result is divided by 10, finally 5 is added to the final result.

Remember that multiplication and division always go before addition and subtraction, unless brackets are used to control the order of operations.

2.2. There's nothing so fickle as a variable

A `variable' is a programming term used to describe a place to store things. The `things' can be numbers, or text, or lists of numbers and text---and all sorts of other items too numerous to go into here. For the moment, let's just think of a variable as something a bit like a mailbox.



You can put things (such as a letter or a package) in a mailbox, just as you can put things (numbers, text, lists of numbers and text, etc, etc, etc) in a variable. This mailbox idea is the way many programming languages work. But not all.

In Python, variables are slightly different. Rather than being a mailbox with things in it, a variable is more like a label which is stuck on the outside of the mailbox. We can pull that label off and stick it on something else, or even tie the label (perhaps with a piece of string) to more than one thing. We create a variable by giving it a name, using an equals sign (=), then telling Python what we want that name to point to. For example:

```
>>> fred = 100
```

We've just created a variable called `fred' and said that it points to the number 100. It's a bit like telling Python to remember that number because we want to use it later. To find out what a variable is pointing at, we can just type `print' in the console, followed by the variable name, and hit the Enter key. For example:

```
>>> fred = 100
>>> print(fred)
100
```

We can also tell Python we want the variable fred to point at something else:

```
>>> fred = 200
>>> print(fred)
200
```

On the first line we say we now want fred to point at the number 200. Then, in the second line, we ask what fred is pointing at just to prove it changed. We can also point more than one name at the same item:

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```

In the code above, we're saying that we want the name (or label) john to point at the same thing fred is pointing to. Of course, `fred' isn't a very useful name for a variable. It doesn't tell us anything about what it's used for. A mailbox is easy---you use a mailbox for mail. But a variable can have a number of different uses, and can point at a whole bunch of different things, so we usually want something more informative as its name.

Suppose you started up the Python console, typed `fred = 200', then went away---spent 10 years climbing Mount Everest, crossing the Sahara desert, bungy-jumping off a bridge in New Zealand, and finally, sailing down the Amazon river---when you came back to your computer, would you remember what that number 200 meant (and what it was for)?
I don't think I would.
I just made it up now, and I have no idea what `fred = 200' means (other than a *name* pointing at the number *200*). So after 10 years, you'll have absolutely no chance of remembering.

Aha! But, what if we called our variable: *number_of_students*.

```
>>> number_of_students = 200
```

We can do that because variable names can be made up of letters, numbers and (_) underscores---although they cannot start with a number. If you come back after 10 years, `number_of_students' still makes sense. You can type:

```
>>> print(number_of_students)
200
```

And you'll immediately know that we're talking about 200 students. It's not always important to come up with meaningful names for variables. You can use anything from single letters (such as `a') to large sentences; and sometimes, if you're doing something quick, a simple and short variable name is just as useful. It depends very much upon whether you want to be able to look at that variable name later and figure out what on earth you were thinking at the time you typed it in.

```
this_is_also_a_valid_variable_name_but_perhaps_not_very_useful
```

2.3. Using Variable

Now we know how to create a variable, how do we use it? Remember that equation we came up with earlier? The one to work out how much money you'd have left at

the end of the year, if you earned $5 a week doing chores, $30 a week on a paper round, and spent $10 per week. So far we have:

```
>>> print((5 + 30 - 10) * 52)
1300
```

What if we turn the first 3 numbers into variables? Try typing the following:

```
>>> chores = 5
>>> paper_round = 30
>>> spending = 10
```

We've just created variables named `chores', `paper_round' and `spending'. We can then re-type the equation to get:

```
>>> print((chores + paper_round - spending) * 52)
1300
```

Which gives the exact same answer. What if you get $2 more per week, for doing extra chores. Change the `chores' variable to 7, then hit the up-arrow key (↑) on your keyboard a couple of times, until the equation re-appears, and hit the Enter key:

```
>>> chores = 7
>>> print((chores + paper_round - spending) * 52)
1404
```

That's a lot less typing to find out that you now end up with $1404 at the end of the year. You can try changing the other variables, then hit the up-arrow to perform the calculation again, and see what effect it has.

```
If you spend twice as much money per week:
>>> spending = 20
>>> print((chores + paper_round - spending) * 52)
884
```

You're only left with $884 savings at the end of the year. This is still only slightly useful. We haven't hit really useful yet. But for the moment, it's enough to understand that variables are used to store things.
Think of a mailbox with a label on it!

2.4. A Piece of String?

If you're paying attention, and not just skimming through looking for the good bits, you might remember I mentioned that variables can be used for all sorts of things---not just numbers. In programming, most of the time we call text a `string'. Which might seem a bit weird; but if you think that text is just `stringing together' (or joining together) a bunch of letters, perhaps it might make a little more sense. *Then again, perhaps it doesn't.*

In which case, all you need to know, is that a string is just a bunch of letters and numbers and other symbols put together in some meaningful way. All the letters, and numbers, and symbols in this book could make up a string. Your name could be a string. So could your home address. The first Python program we created in Chapter **??**, used a string: `Hello World'.

In Python, we create a string by putting quotes around the text. So we can take our useless fred variable, and put a string inside it like this:

```
>>> fred = "this is a string"
```

And we can see what's inside the fred variable, by typing print(fred):

```
>>> print(fred)
this is a string
```

We can also use single-quotes to create a string:

```
>>> fred = 'this is yet another string'
>>> print(fred)
this is yet another string
```

However, if you try to type more than one line of text for your string using a single quote (') or double quote ("), you'll get an error message in the console. For example, type the following line and hit Enter, and you'll get a fairly scary error message similar to the following:

```
>>> fred = "this is two
  File "<stdin>", line 1
    fred = "this is two
                      ^
SyntaxError: EOL while scanning string literal
```

We'll talk more about errors later, but for the moment, if you want more than one line of text, you can use 3 single quotes:

```
>>> fred = '''this is two
... lines of text in a single string'''
```

Print out the contents to see if it worked:

```
>>> print(fred)
this is two
lines of text in a single string
```

By the way, you'll see those 3 dots (...) quite a few times when you're typing something that continues onto another line (like a multi line string). In fact, you'll see it a lot more as we continue.

2.5. Tricks with Strings

Here's an interesting question: what's 10 * 5 (10 times 5)? The answer is, of course, 50.
All right, that's not an interesting question at all.

But what is 10 * 'a' (10 times the letter a)? It might seem like a nonsensical question, but here's the answer from the World of Python:

```
>>> print(10 * 'a')
aaaaaaaaaa
```

This works with more than just single character strings:

```
>>> print(20 * 'abcd')
abcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcd
```

Another trick with a string, is embedding values. You can do this by using %s, which is like a marker (or a placeholder) for a value you want to include in a string. It's easier to explain with an example:

```
>>> mytext = 'I am %s years old'
>>> print(mytext % 12)
I am 12 years old
```

In the first line, the variable mytext is created with a string containing some words and a placeholder (%s). The %s is a little beacon saying ``replace me with something'' to the Python console. So on the next line, when we call print(mytext), we use the % symbol, to tell Python to replace the marker with the number 12. We can reuse that string and pass in different values:

```
>>> mytext = 'Hello %s, how are you today?'
>>> name1 = 'Joe'
>>> name2 = 'Jane'
>>> print(mytext % name1)
Hello Joe, how are you today?
>>> print(mytext % name2)
Hello Jane, how are you today?
```

In the above example, 3 variables (mytext, name1 and name2) are created---the first includes the string with the marker. Then we can print the variable `mytext',

and again use the % operator to pass in variables `name1' and `name2'. You can use more than one placeholder:

```
>>> mytext = 'Hello %s and %s, how are you today?'
>>> print(mytext % (name1, name2))
Hello Joe and Jane, how are you today?
```

When using more than one marker, you need to wrap the replacement values with brackets---so (name1, name2) is the proper way to pass 2 variables. A set of values surrounded by brackets (the round ones, not the square ones) is called a *tuple*, and is a little bit like a list, which we'll talk about next.

2.6. Not quite a shopping list

Eggs, milk, cheese, celery, peanut butter, and baking soda. Which is not quite a full shopping list, but good enough for our purposes. If you wanted to store this in a variable you could create a string:

```
>>> shopping_list = 'eggs, milk, cheese, celery, peanut butter, baking soda'
>>> print(shopping_list)
eggs, milk, cheese, celery, peanut butter, baking soda
```

Another way would be to create a `list', which is a special kind of object in Python:

```
>>> shopping_list = [ 'eggs', 'milk', 'cheese', 'celery', 'peanut butter',
... 'baking soda' ]
>>> print(shopping_list)
['eggs', 'milk', 'cheese', 'celery', 'peanut butter', 'baking soda']
```

This is more typing, but it's also more useful. We could print the 3rd item in the list by using its position (called its index position), inside square brackets []:

```
>>> print(shopping_list[2])
cheese
```

Lists start at index position 0---so the first item in a list is 0, the second is 1, the third is 2. That doesn't make a lot of sense to most people, but it does to programmers. Pretty soon, when you walk up some stairs you'll start counting with zero rather than one. That will really confuse your little brother or sister.

We can show all the items from the 3rd item up to the 5th in the list, by using a colon inside the square brackets:

```
>>> print(shopping_list[2:5])
['cheese', 'celery', 'peanut butter']
```

[2:5] is the same as saying that we are interested in items from index position 2 up to (but not including) index position 5. And, of course, because we start counting with 0, the 3rd item in the list is actually number 2, and the 5th item is actually number 4. Lists can be used to store all sorts of items. They can store numbers:

```
>>> mylist = [ 1, 2, 5, 10, 20 ]
```

And strings:

```
>>> mylist = [ 'a', 'bbb', 'ccccccc', 'ddddddddd' ]
```

And mixtures of numbers and strings:

```
>>> mylist = [1, 2, 'a', 'bbb']
>>> print(mylist)
[1, 2, 'a', 'bbb']
```

And even lists of lists:

```
>>> list1 = [ 'a', 'b', 'c' ]
>>> list2 = [ 1, 2, 3 ]
>>> mylist = [ list1, list2 ]
>>> print(mylist)
[['a', 'b', 'c'], [1, 2, 3]]
```

In the above example, a variable called `list1' is created with 3 letters, `list2' is created with a 3 numbers, and `mylist' is created using list1 and list2. Things can get rather confusing, rather quickly, if you start creating lists of lists of lists of lists... but luckily there's not usually much need for making things that complicated in Python. Still it is handy to know that you can store all sorts of items in a Python list.
And not just your shopping.

Replacing items

We can replace an item in the list, by setting its value in a similar way to setting the value of a normal variable. For example, we could change celery to lettuce by setting the value in index position 3:

```
>>> shopping_list[3] = 'lettuce'
>>> print(shopping_list)
['eggs', 'milk', 'cheese', 'lettuce', 'peanut butter', 'baking soda']
```

Adding more items...

We can add items to a list by using a method called `append'. A method is an action or command that tells Python that we want to do something. We'll talk more about methods later, but for the moment, to add an item to our shopping list, we can do the following:

```
>>> shopping_list.append('chocolate bar')
>>> print(shopping_list)
['eggs', 'milk', 'cheese', 'lettuce', 'peanut butter', 'baking soda',
'chocolate bar']
```

Which, if nothing else, is certainly an improved shopping list.

...and removing items

We can remove items from a list by using the command `del' (short for delete). For example, to remove the 6th item in the list (baking soda):

```
>>> del shopping_list[5]
>>> print(shopping_list)
['eggs', 'milk', 'cheese', 'lettuce', 'peanut butter', 'chocolate bar']
```

Remember that positions start at zero, so shopping_list[5] actually refers to the 6th item.

2 lists are better than 1

We can join lists together by adding them, as if we were adding two numbers:

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 4, 5, 6 ]
>>> print(list1 + list2)
[1, 2, 3, 4, 5, 6]
```

We can also add the two lists and set the result to another variable:

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 4, 5, 6 ]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 5, 6]
```

And you can multiply a list in the same way we multiplied a string:

```
>>> list1 = [ 1, 2 ]
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

In the above example, multiplying list1 by five is another way of saying ``repeat list1 five times''. However, division (/) and subtraction (-) don't make sense when working with lists, so you'll get errors when trying the following examples:

```
>>> list1 / 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

or:

```
>>> list1 - 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'type' and 'int'
```

You'll get a rather nasty error message.

2.7. Tuples and Lists

A tuple (mentioned earlier) is a little bit like a list, but rather than using square brackets, you use round brackets---e.g. `(' and `)'. You can use tuples in a similar way to a list:

```
>>> t = (1, 2, 3)
>>> print(t[1])
2
```

The main difference is that, unlike lists, tuples can't change, once you've created them. So if you try to replace a value like we did earlier with the list, you'll get another error message:

```
>>> t[0] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'tuple' object does not support item assignment
```

That doesn't mean you can't change the variable containing the tuple to something else. For example, this code will work fine:

```
>>> myvar = (1, 2, 3)
>>> myvar = [ 'a', 'list', 'of', 'strings' ]
```

First we create the variable myvar pointing to a tuple of 3 numbers. Then we change myvar to point at a list of strings. This might be a bit confusing at first. But think of it like lockers in a school. Each locker has a name tag on it. You put something in the locker, close the door, lock it, then throw away the key. You then peel the name tag off, wander over to another empty locker, and stick something else in that (but this time you keep the key). A tuple is like the locked locker. You can't change what's inside it. But you can take the label off and stick it on an unlocked locker, and then put stuff inside that locker and take stuff out---that's the list.

2.8. Things to try

In this chapter we saw how to calculate simple mathematical equations using the Python console. We also saw how brackets can change the result of an equation, by controlling the order that operators are used. We found out how to tell Python to remember values for later use---using variables---plus how Python uses `strings' for storing text, and lists and tuples, for handling more than one item.

Exercise 1

Make a list of your favourite toys and name it toys. Make a list of your favourite foods and name it foods. Join these two lists and name the result favourites. Finally print the variable favourites.

Exercise 2

If you have 3 boxes containing 25 chocolates, and 10 bags containing 32 sweets, how many sweets and chocolates do you have in total? (Note: you can do this with one equation with the Python console)

Exercise 3

Create variables for your first and last name. Now create a string and use placeholders to add your name.