Chapter 1

Getting Ready to Program

Python is, itself, a program you run on your computer, which interprets the programs you enter. This chapter takes you through the process of setting your computer up to use Python, and some initial examples to show you what Python can do.

Section 1

How This Book Works

SUMMARY

1. ***Text to type***
2. *Text you see on the screen*
3. **<key> to press**
4. **Example code at**
5. **Chapter ? contains a review of concepts**

Conventions

I tried to write this book so it would be understandable to someone learning programing for the very first time. If you already know a little bit of programming from somewhere, you could also use this book to learn Python, though you may find some of it a little simple. To add a little spice, you will find occasional boxes that look like this: box, with notes for more advanced readers. If you're a beginner, you can ignore these.

There are a few basic conventions we use in this book that it's important to be familiar with, so you find the process fun, not frustrating. Most people will find these things pretty obvious, but let's just make sure everyone's on the same page. First, if you see ***`text that looks like this`***, you are supposed to (or can if you want) type exactly what you see into the computer. Generally you should press <enter> at the end of each line. When you see something between <> characters, this represents a key you are supposed to press, such as <p>, <space> or <enter>. Please note also that <enter> is the same as <return>. Different keyboards give it different names. Sequences like <ctrl-c> mean you should hold down the 'ctrl' key and press c. Text *`like this`* is text you should expect to see displayed on the screen.

The first chapter will explain how to install and run Python, and then give you a series of little examples just to demonstrate some of Python's capabilities, and to give you a flavor of what's to come. Later in the book, we will take apart some of these examples and see how they do what they do.

Too much Typing

This book has quite a lot of code (programs) that you need to type in. You will get the most out of the book if you actually go through the examples and type them in when you see them. In fact, don't be shy, go ahead and play around. Change the examples, and see what happens. It's pretty hard to do any serious harm with anything we're doing in this book. If we're playing with anything really risky, like deleting files from the hard drive, I'll warn you in advance. For the most part, however, if you find that you are playing around and things get completely messed up, you can just exit Python (you'll know what this means later), and start it up again from scratch.

If you're feeling lazy (or type really slowly), there is one alternative. Most of the significant chunks of code in this book are all assembled into one neat package at

Concepts

This book introduces new concepts gradually as we need them in the various fun little projects. The fun projects are presented in order such that we can introduce new concepts a steady, but hopefully not overwhelming, pace throughout the book. However, that means if you wanted to use this book as a reference, it could be challenging to remember exactly where a new idea was introduced. For this reason, we have Chapter ?, which contains a review of the concepts in the book, complete with additional examples. If you find a concept too difficult when it's first introduced, check this chapter for more examples that may help clear up your confusion. This chapter will also serve as a good reference after you finish the book. Also note that there are plenty of good resources on the web for learning about specific concepts if you still find something hard to grasp.

Section 2

Installing Python

SUMMARY

1. **Python installation is platform-dependent (Linux, Mac, Windows).**
2. **This book will use Python 2.7.x. Most of what is covered will also apply to other versions.**
3. **We also install some useful Python libraries which aren't part of the standard distribution.**
4. **Wherever text appears in *`this font`*, this is an indication of something you are expected to type into your computer.**

The Good

Python is, quite simply, a fun language to use. Whereas many programming languages force you to do any specific task one specific way, and make you carefully define every aspect of your program before you can actually do anything, Python is very relaxed, and free-form. For any given task, it is generally possible to come up with a half-dozen different ways to accomplish it in Python. While it permits you to be very rigid in your software design, it also gives you the freedom to simply play around (which is what we will spend most of this book doing).

Python is widespread enough that it is included as a standard part of most (but not all) modern operating systems. The specific version of Python you will have will vary with how old your OS is. The exercises in this book will focus on Python 2.7.x. The vast majority of what we cover will also be valid for earlier versions of Python. Python 3.x has also been available now for some time, however adoption has been slow because it isn't 100% compatible with earlier versions. We will try to mention any places where there is a critical difference between Python 3 and Python 2.

The Bad

Python is what is known as an interpreted programming language. When you write a program in a language like C++ or Fortran, your program is first passed through a **compiler**, to produce an **executable**. This executable runs directly on the CPU

of the computer, and is very fast. In Python, your program just runs immediately, without a compiler. However, since it hasn't been compiled, it may run noticeably slower than the same program written in a language like C++. On the bright side, Many of the libraries of functions provided as a standard part of Python ARE compiled, and run at full speed. So, Python is often used as a **scripting language** to make other, faster programs and libraries do precisely what you want them to.

The Ugly

Many professional programmers, particularly those trained in formal C++ and Java programming, dislike Python's free-form style. They claim it encourages bad programming habits. To some extent, they are correct. If you were writing the software system for a bank, where everything had to work exactly according to specific rules, and 50 programmers all had to work together to produce one gigantic piece of code, you, too, might be fond of rigid rules and guidelines. In such situations, if one guy decides to do things 'their own way', the next thing you know, someone's bank account has accumulated an extra million dollars due to a programming error. Python can, and has, been used for very large projects, but it really shines in situations where a lone programmer is trying to get something done quickly.

Installing Python

Mac Users

Luckily for users of Macintosh computers, Python comes preinstalled with the operating system. There are even a number of Mac-specific libraries which allow you to write complicated programs making use of standard Mac tools. The version of Python you have will depend on what release of OS-X you have installed. Lion (10.7) and Mountain Lion (10.8) include Python 2.7. Earlier versions of OS-X include earlier versions of Python. If you open a **terminal window**, and type *`python --version`*, you will see what you have. The majority of the exercises in this book do not specifically require Python 2.7.

Windows Users

Python will not be preinstalled on windows, however, a windows installer for virtually any version of Windows is available from www.python.org. Simply download the appropriate (Python 2.x) installer and run it. On Windows, you can launch python in two different ways, which we'll talk about later.

Linux Users

Like Mac users, you too are in luck, as virtually all Linux distributions come with Python preinstalled. Open a terminal window and try typing ***`python --version`***. If you see a version number displayed (and it's 2.x), then you're all set. If not, you will need to run the software manager for whichever version of Linux you're using (generally on a system menu) and install Python. Often you will have a choice between Python 2.x and Python 3.x. Again, in this book we will focus on Python 2.x.

IPython

In addition to the standard interactive mode that comes with Python, there is an open source project called IPython, which gives you an interactive mode with some additional capabilities. If you are feeling more adventurous, you may consider downloading and installing IPython in addition to the normal python interpreter, and using ***`ipython`*** rather than ***`python`***, when prompted. Some of the capabilites of IPython will be discussed later in the book, but if you go this route, for the most part you'll need to read the manual to sort out how to use some of its advanced features. If you get frustrated easily, and are new to programming, you may want to hold off on trying IPython for the moment.

iPad Users

Originally Apple didn't permit programming languages on its iDevices, however sometime in 2011 they reversed this policy, and you can now purchase at least one version of Python for the iPad/iPhone. However, it may be challenging to continuously switch back and forth between this book and your iPad, and a few things we'll do will not be possible on the iPad due to its security restrictions, so my advice would be to read the book on the iPad and practice programming on the computer, but it's completely up to you. It is also worth noting that, at the time of this writing, turtle graphics is one of the thing the Python interpreter on the iPad doesn't support, and we use this for a number of the more entertaining early examples.

Section 3

Taking Python Out for a Spin

SUMMARY

1. **You can start Python by typing *`python`* at the command-prompt.**
2. **Python can be used to do basic math like a calculator, for example *`2*5+10`*. If you need scientific functions, like sqrt() or cos(), first you have to type: *`from math import *`***
3. **A string can be created by surrounding text with double quotes, such as : *`"a test"`*. You can also perform addition and multiplication with strings.**
4. **Python has built-in Turtle graphics, which can be used to do simple drawing operations. This emulates a real Turtle robot drawing with a pen.**

Starting Python

There are two fundamentally different ways you can use an **interpreted language** like Python. First, you can use a text editor to create a file containing your program, then you can run the program just like you do any other application on your computer. Alternatively, you can run Python in **interactive mode**, and just type commands into it one after another. It will immediately respond to each command. We will make use of both methods in this book. However, we will begin with the interactive mode, and use this for many of the simple exercises in the book.

On any of the three computer platforms we cover, Python can be run by opening a command prompt, and typing ***`python`***. While you can start it using an Icon on most platforms as well, there are some reasons not to do it that way just yet.

So, go ahead and give it a try. Once you enter ***`python`***, you should receive a prompt, looking something like:

```
odd-2% python
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

>>> is the Python prompt. If you opted to use IPython instead of Python, you will see a prompt like *`In [1]:`* instead, but it has exactly the same meaning. Either way, this is the place where you type all of the nifty Python commands we'll be learning in the interactive exercises.

Your First Python Commands

Next chapter we will start learning Python properly, but let's get started with a few quick examples showing you some easy things you can do.

Python as a Calculator

This is where almost any introduction to Python starts, mainly because it's easy, and can be useful. At the prompt, type: ***1+1*** and press <enter>.

If everything is working as it should, you should see '2' followed by another prompt. Cool, huh ? Ok, ok, perhaps that was a little simple. How about something a little more complicated. Try this:

```
for i in range(10):
    print i,i*i,i*i*i
```

This one is a little trickier. Note that the second line is indented. This indentation is critical to python, as code that is indented the same amount will be executed together (we'll discuss this more in the next chapter). For now, just make sure you either use one or more spaces or a <tab> character to indent the second line.

You'll also note that after you enter the first line, the prompt will change from *>>>* to **`...`** . This means Python is waiting for you to complete a command you didn't finish on the first line. After you enter the command on the second line, you will see another **`...`** prompt. At this prompt you will need to press <enter> again on an empty line to let Python know you don't have any more commands to give.

Whew, quite a long explanation for 2 lines of code, huh ? Don't worry, things will get easier once you learn a few of these simple rules. If you typed everything correctly, you should have seen:

```
0 0 0
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
```

That is, the numbers from 0 - 9 and each number squared and cubed. Still not rocket science, but easier than doing the same thing with your pocket calculator.

Just to get it out of the way, let's list the basic mathematical **operators** in python:

a+b	add	a-b	subtract
a*b	multiply	a/b	divide
-a	negate	a%b	remainder
a**b	a to the b power	pow(a,b)	a to the b power

Those take care of all of the basic math you'll need to do. We'll get to more complicated math later on.

Floating Downstream

There are a few oddities when you first start programming, which may seem a bit bizarre. Enter the following: ***5/2***

Most people would probably expect this to produce the number *2.5*, or perhaps if you don't know much about programming, *2 1/2*. You normally wouldn't expect what you actually get, 2. So, what's going on here ?

As it happens, for reasons we probably shouldn't get into at this point, computers divide numbers into two different types: integers, and **floating point** numbers. We'll save the details for a later chapter, but for the moment, try typing ***5.0/2.0***, and you'll get the general idea.

Fractions

Many people grow up hating fractions, and for many parents, their kids confronting them with homework involving fractions is their worst nightmare. Now, while this isn't true for your typical programmer, it's nice to know that Python has your back. Give this a try:

```
from fractions import Fraction
Fraction(3,45)
Fraction(2,9)*Fraction(3,4)+Fraction(1,2)
```

You'll note that it automatically simplifies the resulting fractions for you.

Mathematical Functions

Now lets try something a little more complex. Let's say we want the square root of a number. Try typing ***sqrt(5.0)***.

Aack! You probably got an error saying it doesn't know what sqrt means. What's up ? Did I misspell sqrt ? No, this is simply our first introduction to modules in Python. All of the math functions live in a **module** called 'math'. So, let's try one more time. Do this:

```
from math import *
sqrt(5.0)
cos(.25)
```

After typing that first line, you suddenly have access to mathematical functions of all sorts. What functions ? Quite a few to choose from. Try this:

```
import math
help(math)
```

This will give you help on the entire math module. Note that after you see the first page of math functions, you get a ***:*** prompt instead of the typical *>>>*. This prompt tells you that python has more than one page of stuff to show you. Press <space> to see the next page, and when you get tired, press <q>.

Strings

Strings ? Musical instruments ? Subatomic particles ? No, in programming, a **string** is a sequence of characters (letters, punctuation, numbers). While math is undeniably important, most of what we do with computers involves words too, so let's see a few simple things we can do with strings.

First, consider a simple string, letters between double quotes:

```
s="This is a string"
print s
```

If you enter this, you will create a variable, ***s***, and print it (the string) on the screen. Great ! We have "This is a string" in a variable. Other than print it out, what can we do with it ?

```
print len(s)
```

Ah ha ! The length of the string ! Useful for some things, but not earth shattering.

```
print s.count("i")
```

We can find out how many "i"s there are in the string. Why ? Well, why not.

```
print "".join(sorted(s))
```

That one is a just a bit less obvious. It sorts the letters in the string in alphabetical order. Probably not the most useful example, but it gives you a feel for some of the possibilities. Later, we'll have a whole chapter on programming for word games. How about math with strings ?

```
"2.0"+"3.0"
```

You should try this one yourself. If you haven't done much programming before, you might guess that this would produce *"5.0"*, when in reality, it produces *"2.03.0"*. So, strings can be added, but with strings this means they should be joined together, not added mathematically. Strings can be multiplied as well:

```
"2"*4
```

This produces the same thing ***"2"+"2"+"2"+"2"*** would, that is *"2222"*.

Of course, there are many more interesting things we can do with strings, but that will have to wait until the next chapter.

Turtles

Before the days of high resolution color monitors, there were simple black and white display, capable of showing only letters and numbers. No graphics at all.

Before this, were the TTYs: Imagine a typewriter connected to a computer. As you typed, each letter was printed on the paper, but was also sent to the computer. The computer's responses were then ghost-typed on the same paper. In these days, the idea of "graphics" on a computer was whatever you could do with letters and numbers on a printed piece of paper. Not only was there no color, but there weren't even lines, aside from the -, |, / and \ characters.

Ok, why am I boring you with history ? Back in the 50's, early robots known as 'turtles' were developed. In the late 70's/early 80's, these turtles were adapted to be educational tools as graphics devices for the computer. The turtle had a pen which could be moved up and down, and the robot could be given simple commands like move forward, turn 10 degrees left, etc. This could be used to draw pictures on paper. While the robots aren't very common any more (though you can still get/make them), the idea is still alive and well. Most of the time the turtles are now little triangles that move around on your computer screen, trailing a line behind them.

Now that our history lesson is over, let's try it:

```
from turtle import *
s=Screen()
goto(0,0)
```

Look ! A window appeared, and when you typed goto(0,0), a little arrowhead (the turtle) appeared in the middle of the window. What fun ! What next ?

```
for i in range(36):
    forward(10)
    left(10)
```

Look, a circle ! (we have to start somewhere, don't we ?) How about something with a little more flair:

```
reset()
goto(-125,-125)
clear()
for i in range(61):
    forward(250)
    left(118)
```

Ever play with a Spirograph before ? You can do interesting things with turtles without much effort. Note that the turtles are intentionally slowed down so they act more like a turtle robot might. Clearly the computer is capable of drawing a lot faster than this.

Example 1.1 What you should see

Section 4

Problems

While you haven't really been 'taught' anything yet, if you're clever, you may be able to figure out the examples you've seen enough to try your hand at a few simple problems. Of course, the answers are provided as well.

Problem 1 - Print the integers from 0 to 10 and the square root of each.

Solution 1.1

Scroll through the images to see the solution.

Problem 2 - Modify the turtle examples and see if you can draw:

a) A hexagon

b) The spirograph example is loosely based on triangles, modify it so its based on squares instead.

Solution 1.2

Again, start by running ***`python`****.*