Python Primer

Patrice Koehl
Department of Computer Sciences, University of California, Davis.

Acknowledgments:

This primer is mostly a compilation of information found in books / web resources that I highly recommend:

- http://docs.python.org/reference/introduction.html ; this is a reference manual, and not a tutorial, but provides invaluable information about the language: do not hesitate to consult it!
- http://wiki.python.org/moin/BeginnersGuide ; a Beginner's guide with many links to resources for writing and running Python programs
- Michael Dawson, "Python programming for the absolute beginner", 2nd edition, Thomson Course Technology, ISBN: 1-59863-112-8; this is the class textbook; a great resource book on Python

Introduction

1. Why Python?

"Python" is an interpreted computer language developed in the 1980s and first released in 1991. Its design philosophy emphasizes **programmer productivity** and **code readability**.

It is important to understand that there is always more than one way to solve a problem. In programming, Python focuses on getting the job done. One Python program may be faster than another, or more concise, or easier to understand, but if both do the same things, there won't be a judgment that defines which one is "better". This also means that you do not need to know every detail about the language to do what you want with it.

Python has strength that makes it an ideal language to learn and use:

- It is completely free, and available on all operating systems
- It is very easy to learn
- Python was designed to be easy for humans to write, rather than easy for computers to understand. Python syntax is more like English than many other programming languages
- Python "talks" texts. It works with words and sentences, instead of characters. Files are series of lines, instead of individual bytes.
- Python is very portable. Python programs can be run on any computers, as long as Python is installed on it.
- Python is a "high-level" language: you do not have to worry about the computer's operation (such as allocation and de-allocation of memory, …).

It is only fair to mention that these strengths can also translate into weaknesses:

- Python takes care for you of all "low-level" operations: this may not always lead to efficient code
- Python is interpreted, and loses the efficiency of compiled languages.
- Python users then write programs for small, specific jobs. These programs are usually for the programmer's eye only, and as such are often incomprehensible to everyone but the original programmer. In that respect, I can only emphasize the need for clarity, as well as for useful comments in your source files!
- Python was designed to be easy for humans. As a consequence, it is relatively lenient on the style you use. This can lead to bad programming habits. As an analogy, think of what would happen to your English writing style if nobody had ever cared about how you write as long as they understand what you have written. *To avoid this, the key is to develop first a method to solve your problem that is independent of Python (or any other language), and then to adapt this method to Python.*

2. What is Python used for?

Python has been successfully implemented in many software applications as a scripting language.

Python is a very useful programming language for web applications.

Python is used widely for game development, for 3D animation packages, in the information security industry,…

3. How do I get Python?

Python has been ported to many platforms, and will certainly run on the standard operating systems such as UNIX, Linux, Solaris, FreeBSD, all flavors of Windows, and Apple MacOS.

Python 2 versus Python 3

In December 2008, the Python consortium released a completely new version of Python, Python 3.0, that is not backward compatible: this means that programs written with Python 1 or Python 2 may not run under Python 3.0. At this stage, we will stay with Python 2, as it remains the most common version found on many operation systems. Even if Python 2 and Python 3 are not fully compatible, once you'll know Python version 2, switching to version 3 will be easy.

Where to get Python:

- You can get the source to the latest stable release of Python from http://www.python.org. Remember that you want Python 2 at this stage.
- Binary distributions for some ports are available at the same address
- You can get binary packages of Python for Linux, Solaris, Mac OS and Windows from ActiveState at http://www.activestate.com/ActivePython (free for download)

Installing Python on Linux/UNIX

Python is freely available, and usually comes packaged with most Linux/UNIX distribution. Type **python** from a shell prompt to check this. If you see something that starts with the text **Python**, then you already have python. If you don't, check the install media (CD or DVD from which you installed Linux): the Python package should be available on it. Otherwise, get the binaries from the site mentioned above.

Installing Python on Mac Os

Again, Python comes packaged with the different flavors of Mac OS X, and you probably have nothing to do! Check it using **python** from a terminal. If you do not have it, I would advise getting it from ActiveState.

Installing Python on Windows

Installing ActivePython is quite straightforward. Download ActivePython's Python installer for Windows from the ActiveState web site (again, it is free for download). Choose the appropriate version for the operating system you have (32 bit, or 64 bit). I would strongly advise using the MSI installer, in which case you will need Windows Installer 2.0+ (which you probably already use). It should work under Vista and Windows 7, but I have not tried it.

4. Getting an IDE for Python

IDEs (integrated Development Environment) are great tools for learning a computer language and use it efficiently. There are many IDEs available for Python: see http://wiki.python.org/moin/IntegratedDevelopmentEnvironments for a list of such IDEs.

I strongly recommend IDLE that is available for nearly all platforms. See http://www.python.org/idle/doc/idlemain.html ; it should come by default when you install Python on Windows; you have to install it however on Linux and MacOS platforms. For the latter, please check: http://challenge.ncss.edu.au/gsg-osx/ .

5. Using Python

You have two main ways to use Python:

- Use it directly through a Control Window: again, I strongly advise using the IDLE interface
- Write the Python program (module) using a text editor, and then execute this program through the python interpreter. Again, this can be done using IDLE; alternatively, you can use standard text editors for this task (see below)

Editing a Python program

Python source code is just plain text and should be written with a plain text editor, rather than a word processor. If you are using Windows, you can use Notepad, despite its annoying tendency

to rename file extension to .txt. You may also use Word, as long as you save the file as text, with line breaks.

I would really recommend getting a good programmer's editor. For Windows and Mac, I can recommend jEdit (http://www.jedit.org/): it is free (open source), runs under Windows, Mac OS X, Unix and Linux. It is easy to use, highly customizable, with many useful plugins. Another option for Mac user is TextWrangler (http://www.barebones.com/products/textwrangler/).

Naming a Python program

Traditionally, UNIX programs take no extension, while Windows files take a three-letter extension to indicate their type (.exe for an executable, .doc for a document –usually Word file-, .xls for a spreadsheet, …); the standard extension for Python program is py.

Obviously, the choice of the name in front of the extension is entirely yours!

Using Python in an IDE

If you are mainly using your computer in a graphical environment like Windows or X, you may not be familiar with using the command line interface, or "shell". The "shell" is the program that gets input from you through the keyboard. The "shell prompt" or just "prompt" refers to the text that prompts you to enter a command. The standard prompt in IDLE is:

>>>

i.e. 3 chevrons.

In this primer, I will use a prompt that looks like:

>>>

I will show the text that you would type in bold and the text the computer generates in italic:

>>> print "Hello world!"
Hello World!

6. Your first Python program

Traditionally, the first program anyone writes in a new language is called "Hello World!", where you make the program prints that statement. Python allows us to do so using the print statement. The simplest form of the print statement takes a single argument and writes it to the standard output, i.e. the command window you have open. So your program consists of the single statement:
print "Hello World!\n"

You can execute this command directly in the IDLE main window, or you can incorporate it into a Python module, hello.py. The file hello.py contains:

```
#
print “Hello World!\n”
#
```

The different elements of a Python script:

- ***Documenting the program***: any line (except the first) starting with a sharp (#) is treated as a command line and ignored. This allows you to provide comments on what your program is doing: this is extremely useful, so use it! More generally, a line in a Python script may contain some Python code, and be followed by a comment. This means that we can document the program “inline”.

- ***Keywords:*** Instructions that Python recognizes and understands. The word **print** in the program above is one example. There are two types of keywords:

    - *functions* (such as the print keyword); these are the verbs of the programming language and they tell python what to do.
    - *Control keywords*, such as if and else.

    The number of Python keywords is small:

and	**del**	**from**	**not**	**while**
as	**elif**	**global**	**or**	**with**
assert	**else**	**if**	**pass**	**yield**
break	**except**	**import**	**print**	**class**
exec	**in**	**raise**	**continue**	**finally**
is	**return**	**def**	**for**	**lambda**
try				

    It is a good idea to respect keywords, and not use them as names in your programs!

- ***Modules:*** Pythons come with a large list of modules that increases its functionality; these modules add keywords to the small list provided above, but are only available when the module has been specifically called. For example, adding:

    use numpy

    adds the modules of numerical functions “numpy” that are now accessible to the programmer.

- ***Statements:*** Statements are the sentences of the program. Python is lenient however, and does not need a full stop to end a statement. The indentation levels of consecutive lines

are used to generate INDENT and DEDENT, which in turn are used to determine the grouping of statements.

- ***White space:*** White space is the name given to tabs, spaces, and new lines. Python is quite strict about where you put white space in your program. For example, we have seen that we use indentation to help show the block structure of statements.

- ***Escape sequences:*** Python provides a mechanism called "escape sequences" to output special characters/actions: the sequence \n in the program above tells Python to start a new line. Here is a list of the more common escape sequences (also called "metacharacters"):

Escape Sequence	Meaning
\t	Tab
\n	Start a new line
\r	Carriage return
\'	Single quote
\"	Double quote
\\	Backslash
\b	Back up one character ('backspace')
\a	Alarm (rings the system bell)

Simple exercises:

1) Write a program printline.py, that prints the sentence "This is my second program":

    a. As a single line
    b. With a single word on each line.

2) Find an online manual for Python

3) Which of the following statements are likely to cause problems:

    a. print "This is a valid statement\n"
    b. print "This is a valid statement"\n
    c. print "This is a "valid" statement"
    d. printx "This is a valid statement\n"