Python 101 (part 1): Snake Eyes

By Vikram Vaswani

Table of Contents

Top Gun.....1

A Reptile By Any Other Name.....2

The Jedi Master Speaks.....3

Start It Up.....5

Dissecting A Python...Program.....8

Milk And Toast And Honey.....10

Adding Things Up.....12

Top Gun

Perhaps it's just my imagination, but Python programmers seem to command a great deal more respect than their Perl or PHP counterparts.

In the consciously−elitist world of software engineering, a developer with a few years of Python under his belt gets the best cubicle, the prettiest girl and the respect of his neighbours; people move out of the way when he strides down the hall, and colleagues turn to him for creative and elegant solutions to the problems they encounter. Walk into a job interview and mention Python when reciting your qualifications; you'll immediately see a glint of recognition in the interviewer's eyes, an awareness that, in the hierarchy of software developers, you're one of the top guns.

I'm exaggerating a little, of course. However, the fact remains that Python, by its very nature, forces most developers to acquire an understanding of object−oriented programming concepts and, by implication, design software that is reusable, extensible and modular. In the long run, this translates into better software engineering, greater code maintainability and less testing − all of which are music to the ears of the average manager.

Over the next few weeks, I'm going to take you on a guided tour of this powerful programming language, and demonstrate some of its capabilities to you. I'll be covering everything you need to know to get started with Python, from using the command−line and understanding Python's data structures, to trapping errors and writing your own Python modules.

If you already know Perl, PHP or Java, many of the concepts explained here may already be familiar to you; however, Python, like its namesake, does add some interesting twists to standard programming constructs, and I'll be sure to point them out to you. If, on the other hand, you're completely new to this, fear not − I'll keep it as simple and non−threatening as I can.

A Reptile By Any Other Name

If you look at a dictionary, a python is defined as "a large snake that crushes its victims."

Python's own Web site, http://www.python.org/, is even more specific:

python, (Gr. Myth. An enormous serpent that lurked in the cave of Mount Parnassus and was slain by Apollo)

1. any of a genus of large, non−poisonous snakes of Asia, Africa and Australia that suffocate their prey to death.

2. popularly, any large snake that crushes its prey.

3. totally awesome, bitchin' language that will someday crush the $'s out of certain other so−called VHLL's ;−)

The brainchild of Guido van Rossum, Python is a powerful and flexible programming language built on strong object−oriented fundamentals. By combining the usability of scripting languages (like Perl and PHP) with the rich feature set of traditional languages (like C++ and Java), it offers an irresistible combination of speed and power. Originally developed in 1989 as part of the Amoeba Project at CWI in the Netherlands, it was released to the user community as freeware, and has become a popular language for high−level application development over the past decade.

In fact, Python's clean syntax and object−oriented framework has helped make it popular among an A−list of "customers", including CNRI, Xerox PARC, Red Hat and Australia's Melbourne Cricket Ground.

The Jedi Master Speaks

Some of Python's most powerful features include:

Object−oriented framework: Built from the ground up as an object−oriented language, Python provides built−in constructs that make it simple for developers to structure code for maximum reusability. Python's "dynamic typing", which automatically recognizes objects like numbers, strings and lists, and negates the need to declare variable types and sizes, offers an advantage not found in languages like C or Java, while automatic memory allocation and management, together with a vast array of pluggable libraries and high−level abstractions, complete the picture.

Extensibility: Python makes it easy to add new capabilities to your code, by importing specialized libraries into your program. A large number of pre−built libraries ship with the standard Python distribution, and more are available free of charge on the Web. Additionally, Python also allows you to compile new "extension modules" for additional functionality, or to improve performance.

Portability: Python distributions are available for a wide variety of operating systems, including Windows, Macintosh, Amiga and most flavours of UNIX, including Linux. This portability ensures that code written on one platform will work on others with zero or minimal changes required. In case a distribution isn't available for a specific platform, you can even download the source code and compile it yourself!

Readability: Python's clear and elegant syntax, and emphasis on proper indentation of code blocks, improves code readability, making it easier to understand code a year later. By imposing lexical and syntactical rules on language constructs, Python reduces the time spent on understanding (and developing) application logic, and also makes program code easier to maintain over time.

Performance: There's an interesting piece of trivia geeks should be aware of here. Python is an odd hybrid of compiled and interpreted languages: the code you write is first converted to so−called "bytecode" and then executed by the Python interpreter. Since executing bytecode is faster than interpreting statements line−by−line, Python offers a substantial performance advantage of other interpreted languages. It should be noted, however, that the conversion to bytecode is handled internally by Python, and is not visible to the end user.

Open licensing model: Like Perl and PHP, Python is available for free, to anyone who wants it, over the Internet, in both source and binary form. Users may use and distribute it without restriction, and even charge a fee for it if they so desire.

Programmers love to debate the virtues of different programming languages, and it's no different here. However, Python comes off favourably in most of these comparisons. You can find detailed evaluations of Python versus other programming languages at http://www.python.org/doc/Comparisons.html. And you might want to take a minute to enjoy this nugget culled from the Python humour pages at http://www.python.org/doc/Humor.html

EXTERIOR: DAGOBAH −− DAY

With Yoda strapped to his back, Luke climbs up one of the many thick vines that grow in the swamp until he reaches the Dagobah statistics lab. Panting heavily, he continues his exercises −− grepping, installing new packages, logging in as root, and writing replacements for two−year−old shell scripts in Python.

YODA: Code! Yes. A programmer's strength flows from code maintainability. But beware of Perl. Terse syntax...more than one way to do it...default variables. The dark side of code maintainability are they. Easily they flow, quick to join you when code you write. If once you start down the dark path, forever will it dominate your destiny, consume you it will.

LUKE: Is Perl better than Python?

YODA: No... no... no. Quicker, easier, more seductive.

LUKE: But how will I know why Python is better than Perl?

YODA: You will know. When your code you try to read six months from now.

Start It Up

With that out of the way, let's get down to actually doing something with Python. I'm assuming here that you've already got Python up and running on your system; if you haven't, now is a good time to visit http://www.python.org/ and download a version for your platform (installation instructions are included with each distribution, and you might also want to download a copy of the Python documentation and function reference.)

The examples in this series will assume Python on Linux, although you're free to use it on the platform of your choice.

If you're on a UNIX system, a quick way to check whether Python is already present on the system is with the UNIX "which" command. Try typing this in your UNIX shell:

```
$ which python
```

If Perl is available, the program should return the full path to the Python binary, usually

```
/usr/bin/python
```

or

```
/usr/local/bin/python
```

There are two primary methods to run Python code − via the command line, or via a script.

The command line interpreter allows you to type in Python code line by line, and executes it immediately; this is a great way to see the result of a particular statement or set of statements. You can start up the interpreter by typing in the full path to the Python binary at any command prompt.

Once the interpreter starts up, you'll see a brief copyright notice and some version information, followed by a ">>>" − this indicates that the interpreter is ready and willing to receive your commands. Here's a brief example of what this might look like:

```
$ python
Python 1.5.2 (#1, Aug 25 2000, 09:33:37) [GCC 2.96 20000731
(experimental)] on
linux-i386
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> print "Snake Eyes"
Snake Eyes
>>>
```

You can exit the interpreter with a Ctrl-D.

The disadvantage of the command line interpreter is that your code is lost once you exit it. Therefore, Python also allows you to save your code to a text file, and run this code as a script from the command line...as the following example demonstrates.

```
# eyes.py - print a line of output
print "Snake Eyes"
```

```
$ python eyes.py
Snake Eyes
```

Python scripts and modules (explained later) typically end in a .py file extension.

In the example above, I've specified the name of the script to run as a parameter on the command line. If you're on a UNIX system, you have the additional option of directly running the script by name, by adding a line specifying the location of the command interpreter at the top of the script file.

```
#!/usr/bin/python

# eyes.py - print a line of output

print "Snake Eyes"
```

This technique should be familiar to anyone who's programmed in Perl or bash.

Note that the script file must have executable privileges − you can accomplish this with the "chmod" command.

```
$ chmod +x eyes.py
```

I'll be using both these techniques to demonstrate code snippets over the course of this tutorial.

Dissecting A Python...Program

Let's take a closer look at the script above. The first line

```
#!/usr/bin/python
```

is used to indicate the location of the Python binary. As I've just explained, this line must be included in each and every Python script, and its omission is a common cause of heartache for novice programmers. Make it a habit to include it, and you'll live a healthier, happier life.

Next up, we have a comment.

```
# eyes.py − print a line of output
```

Comments in Python are preceded by a hash (#) symbol. If you're planning to make your code publicly available on the Internet, a comment is a great way to tell members of the opposite sex all about yourself − try including your phone number for optimum results. And, of course, it makes it easier for other developers (not to mention you, circa 2010) to read and understand your code.

And finally, the meat of the script:

```
print "Snake Eyes"
```

In Python, a line of code like the one above is called a "statement". Every Python program is a collection of statements, and a statement usually contains instructions to be carried out by the Python interpreter. In this particular statement, the print() function has been used to send a line of text to the screen. Like all programming languages, Python comes with a set of built−in functions − the print() function is one you'll be seeing a lot of in the future. The text to be printed is included within single or double quotes.

Just as you print text, you can also print numbers...

```
Python 1.5.2 (#1, Aug 25 2000, 09:33:37) [GCC 2.96 20000731
(experimental)] on
```

```
linux-i386
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> print 4
4
>>> print 3443634
3443634
>>>
```

...or the results of numeric calculations.

```
>>> print 24+1
25
>>> print 12*10
120
>>>
```

Although every Python statement usually begins on a new line, a single statement can be split across lines with a backslash (\). It is not necessary to end each statement with a semicolon (;), as in Perl and PHP.

```
>>> print \
... "Snake Ey\
... es"
Snake Eyes
>>>
```

Milk And Toast And Honey

Like every programming language worth its salt, Python allows you to assign values to variables, the fundamental building blocks of any programming languages. Think of a variable as a container which can be used to store data; this data is used in different places in your Python program.

A variable can store both numeric and non−numeric data, and the contents of a variable can be altered during program execution. Finally, variables can be compared with each other, and you − the programmer − can write program code that performs specific actions on the basis of this comparison.

The manner in which variables are assigned values should be clear from the following example:

```
>>> alpha = 99
>>> print alpha
99
>>> beta = 2
>>> print beta
2
>>> gamma = "Milk and toast and honey"
>>> print gamma
Milk and toast and honey
>>>
```

Although assigning values to a variable is extremely simple − as you've just seen − there are a few things that you should keep in mind here:

* Every variable name must begin with a letter or underscore character (_), optionally followed by more letters or numbers − for example, "a", "data123", "i_am_god"

* Case is important when referring to variables − in Python, a "cigar" is definitely not a "CIGAR"!

* The equals (=) sign is used to assign a value to a variable.

* It's always a good idea to give your variables names that make sense and are immediately recognizable − it's easy to tell what "net_profit" refers to, but not that easy to identify "np".

* Unlike Java and C, Python does not require you to declare the type of variable prior to assigning it a value. It's behaviour here is closer to PHP, which allows you to assign any type of value to a variable without declaring it first.

* Also like PHP, variables are created when they are assigned values − it is not necessary to declare them first.

* Finally, Python variable names are not preceded with a $ sign, unlike most of its counterparts. Once you get

used to it, you'll find that this actually adds to readability.

Adding Things Up

Here's another program, this one performing an arithmetic operation on a variable and displaying the result.

```
>>> num1 = 10
>>> num2 = 5
>>> num3 = num1 + num2
>>> num3
15
>>>
```

Note that Python also allows you to display the value of a variable (in the command line interpreter only) by stating the variable name without a preceding print() function call.

Finally, Python also allows you to simultaneously assign a value to multiple variables − check it out.

```
>>> huey = dewey = louie = 75
>>> dewey
75
>>> louie
75
>>>
```

And that's about it for this introduction to Python. In this article, I've provided a brief history of Python, and explained some of the features which make it so popular among discerning developers. I've also discussed the two methods available to run a Python program, demonstrated the print() function, and offered some insight into the rules governing Python variables.

In the next article, I'll be delving deeper into Python's data structures and built−in objects − numbers and − together with a closer look at the various arithmetic and comparison operators. See you then!