Python superbrief overview

There are currently two major branches of Python, 2.x and 3.x. The 3.x branch doesn’t add anything very significant to the language, and is more to clean up a few things that were seen as inconsistent. However, code is not backwards-compatible from 3.x to 2.x. So if you download the 3.x interpreter and attempt to run a module from a project written in 2.x, you may have problems.

There is a means of converting 2.x to 3.x code, but there’s nothing bad about sticking to 2.x. If you’re not sure which one to grab, I would grab 2.x just to be sure that you’ll be compatible with a higher percentage of modules out there.

Python is an interpreted language, and does not produce platform-specific executables. Your download of Python will include its interpreter and an “IDE” called IDLE. IDLE isn’t a full-blown IDE, but it’s a great place to play around with Python code and test things out. It’s a shell where you can throw Python commands at the interpreter and check results.

There are a bunch of Python IDEs out there. I use either Notepad++ or PyDev which is a full-featured add-on for Eclipse. There are a ton of other options including WingIDE and Kommodo IDE.

List of Python IDEs and discussions:

<http://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

Python is object-oriented, but can be written like a script (for instance, you don’t have to define a class just to be the container for a Main method where execution starts). Python reads code from top-to-bottom, left to right.

Python is strongly typed, meaning that it cares about what types your objects are. For instance, if you attempt to concatenate an integer to a string, it’s going to yell at you for not explicitly casting that integer. It’s also dynamically typed, meaning that you can reassign variables at runtime without any problems. The secondary effect of this is that you don’t have to define variables, you can simply say “i=5”. This could potentially cause problems, since type errors will be caught at runtime instead of at compile time, but as long as you know that you have to be careful with types going in, you should be ok.

Useful functions:

dir() - the parameter to this function will be an object you wish to inspect. It returns a list of the available functions for the object

type() – the parameter for this function is an object which you want to know the type of. It’s better to write code that doesn’t need to you keep checking types. If you start off every re-used function with a check to see if you received the object type that you’re expecting, then you’re doing something wrong (so sayeth the Pythonistas).

help() – If you issue this command without a parameter from the interpreter command line, it puts you in help mode where you can get quick explanations for most of the language keywords and features. Putting a parameter with a descriptor brings you targeted help.

Everything in Python is “by ref”, except for the immutable types (tuples, ints, strings, etc). If you pass an object into a function and change it, it changes in the calling code. If you want to make a copy of an object, you can use deepcopy:

from copy import deepcopy

a = deepcopy(b)

Python uses whitespace to delimit code blocks. The first time you indent (to show a separate code block), Python remembers this indent standard and expects the rest of the code to use it. Using whitespace as a code block delimiter eliminates the need for curly braces, and forces good coding habits.

You do not have to end a Python statement in a semicolon unless you want to put multiple commands on one line.

Importing modules can be done in four ways:

**from** name **import** **\*** 🡨 (this imports every function name from the module directly into your current namespace. Not advised to do this)

**from** name **import** function 🡨 (this imports one function into your namespace)

**import** name 🡨 (imports the module itself; you can then call functions off of it using name.function)

name **= \_\_import\_\_(‘**name**’)** 🡨 (imports using a string; the advantage here is that you can use a variable to represent the name of the module you want to import)

The three main import uses with examples:

import math

math.sqrt(9)

----

from math import sqrt

sqrt(9)

----

from math import \* # However, this imports EVERYTHING into your namespace. Not good!

sqrt(9)

Python supports all of the basic types you’d expect – integers, strings, floating point variables, characters, etc… Two data types you will use extensively are Lists (dynamic arrays) and Dictionaries (hashmaps). Python is extremely expressive in terms of working with these types.

**See code samples for examples of List usage**

The amount of useful modules out there for Python is what makes it great.

PyGame : SDL wrapper similar to how XNA is DirectX wrapper. Graphics/sounds/input

Py2Exe: for building platform-specific executables

Beautiful Soup – for manipulating, stripping, replacing or otherwise working with markup text

SciPi + NumPy – functions for use in science and statistics/numerical analysis

Iron Python - .net implementation of Python; uses the .net class library and compiles down to runnable .net code.

Jython – Java implementation of Python; uses the Java class library and compiles down to runnable java bytecode

wxPython / pyQt – UI frameworks

The built-in Python modules provide an incredible array of functionality.

Database access, working with audio files, file compression, xml parsing, multithreading, object serialization, working with images, server functions, encryption, hashing, FTP, email, working with cookies, etc…

There are thousands of great projects out there. Chances are, if you need something for a program you’re writing, someone has a module out there that will make your life easier.