Companion Notes for Python Programming Tutorial at *www.learnpython.org*

by Sheila Kannappan

These notes accompany the first 14 sections of the tutorial at *http://www.learnpython.org/*. However, we will not use the online Input/Output boxes. Rather, we will write code with **spyder** locally using python 3. You may simply copy/paste each command of the online tutorial into the file editor on spyder, overwriting the previous step and running it. To get set up, open your spyder program as done in the previous tutorial.

Learn the Basics

1. Hello, World!
2. Variables and Types
3. Lists
4. Basic Operators
5. String Formatting
6. Basic String Operations
7. Conditions
8. Loops
9. Functions
10. Classes and Objects
11. Dictionaries
12. Modules and Packages

Advanced Tutorials

13. Generators
14. List Comprehensions

Why two versions of python?? Unfortunately, you are learning python at a time of transition. The creator of python decided to fix some issues with the language in python 3 in such a way that *it is not backwards compatible.* The changes were planned for a while, and the last version of the earlier family of python releases, python 2.7, is designed to make the changeover as painless as possible, but nonetheless two things are true:

1) all future development will follow the lineage and new syntax of python 3, as the previous lineage has been frozen at 2.7
2) many existing packages have not yet been converted to python 3 (even one we've provided, Fabric) and must be run under python 2.7

Fortunately, only a few changes will affect beginning users – chiefly, the syntax of the *print* command and the syntax for integer vs. floating point division. Visit the following webpages for all the gory details:

http://wiki.python.org/moin/Python2orPython3
http://docs.python.org/3/whatsnew/3.0.html
http://python3porting.com/preparing.html

Now please complete the tutorial sections sequentially, noting the clarifying comments/extra exercises given for selected sections below. Note: since white space has syntactical meaning in python, *never* use tabs in your python code!

1. Hello, World!

In the ipython console in spyder, print out the phrase "Hello world!". Notice the use of comments (anything in your code that has a # in front of it) that are not actually executed.

In python 2.7, you used to be able to print by just typing *print "Hello World!".* Try this in your ipython terminal (which is running python 3 or greater). Did it work?

2. Variables and Types

If you don't know the phrase "object oriented" don't worry! It will become clearer later.

4. Basic Operators

Python 2.7 performs division differently than Python 3+. In Python 3, dividing 5 by 2 will return a float (2.5), while diving 5 by 2 in Python 2 will return an integer (2).

7. Conditions

The "is" operator gets at the idea of an "object" in object-oriented programming. The idea is that a particular variable has an identity (perhaps it would be helpful to imagine it at a specific location on a hard disk) which is independent of its value. The "id" function helps clarify matters, because in general any two variables a and b will return different id's, even if they have the same value, unless b was *defined* by b=a, in which case b has no identity independent of being the same as a. In python, a or b could even be functions, so "object" is a very general term for any defined entity in your program. A potential confusion arises because small integers from -5 to 256 are hardwired as objects in python before you ever use them in your code. Therefore we get:

```
>>> a = 256
>>> b = 256
>>> a is b
True # both were linked to the hardwired object "256" >>> a =
257
```

```
>>> b = 257
>>> a is b
False # "257" was not an object, so a and b are distinct
```
To understand this weirdness, look at the id's:

```
>>> a = 256
>>> b = 256
>>> id(a)
9987148
>>> id(b)
9987148
>>> a = 257
>>> b = 257
>>> id(a)
11662816
>>> id(b)
11662828
```

Bottom line, if a "is" b, then id(a) == id(b).

8. Loops

An iterator generator can only be used in a loop – it doesn't make sense outside that context because it is designed to suspend execution until a "for" statement comes back and asks it for another number. To see the difference compared to a standard list, we'll use python interactively. In the ipython terminal, type *print(range(0,7))* and then *print(list(range(0,7))* and compare the results.

12. Modules and Packages

A lot of the exercises in this section aren't meant to be ran, but instead are an outline of how modules would be utilized if you were running your own modules. Only a few of the examples near the bottom are actually expected to run, so be mindful of that.

The command written `>>> dir(urllib)` should say *print(dir(urllib))*.

The sentence `"This file can be empty, and it indicates that the directory it contains is a Python package"` should say `"This file can be empty, and it indicates that the directory it is contained in is a Python package".`

API = "applications programming interface" = functions the package provides for use by default

13. Generators

Note the generator doesn't get looped over, just paused/suspended.

For now you should stop after tutorial #14. Obviously we've just scratched the surface. If you want to dig deeper on your own, besides finishing the online tutorial, you might look into any of these links:

Brief intro to scientific computing with specific python packages:
www.physics.unc.edu/~sheila/emmettpythonpresentation.pdf
Google course on python:
http://code.google.com/edu/languages/google-python-class/
Translation tables between common programming languages:
http://mathesaurus.sourceforge.net/
astronomy-specific resources:
http://stsdas.stsci.edu/perry/pydatatut.pdf