The grand introduction

When I started to learn Python in 2012 with the aim of doing Astronomy, I didn't find any resources to be imminently helpful. Everything was either too advanced or a computer science style tutorial that went way too slow. It is my hope that after this very fast-paced course you are able to do some Astronomy, with some idea of what's going on under the hood. Don't hesitate to use this as a reference, but do not rely on it. Consult web services like stackoverflow or /r/learnpython when possible as they will have up-to-date answers for your questions.

So you want to learn Python?

Before I get into any of the meat here I want to clear up a few misunderstandings or misconceptions:

- Python is slow .. This mostly a product of a common misunderstanding about computer languages. Languages aren't slow, implementations are. It's very easy to shoot yourself in the foot
- Python 3 is a whole new language, Python 2 is the way to go .. This is a dirty lie. The differences for scientists between the two major versions are trivial, and Python 3 is supported by every major package you'll need. Python 3 was released in 2008 and has been supported by numpy since 2010, and all other major numerical packages followed suit within a year or so. Especially if you're a newcomer to the language there is no good reason to default to Python 2.

Setting up for some programming

You'll need two tools to start writing some Python: a text editor or Integrated Development Environment (IDE) and a Python runtime. There is a wonderful product by Continuum Analytics called Anaconda that contains both these things and more tools you'll need later. At the end of the installation you should be asked if you want to prepend something to your PATH variable. You want to do that thing. Head over to https://www.continuum.io/downloads and download the latest installer for your system.

Anaconda is a Python distribution. It contains the CPython interpreter which will run your Python code, a bunch of packages for Python (that can be nigh impossible to install on Windows), and a nice IDE called Spyder. Anaconda also offers to prepend an entry to your PATH variable, which ensure your new installation will work without hiccups and without learning about virtual environments.

From here on I will assume that the reader is using Spyder and Python 3.5+.

A first program

When you open Spyder you'll be presented with an editor on the left and some consoles on the right. The editor already contains a text file with some contents; it specifies that its contents are to be treated as unicode and contains a bit of documentation. Neither of these are important to what we're doing. Enter the simple line `print('Hello world')` into the editor. You now have a simple Python script that you can run. The simple green arrow in the toolbar, the `Run` dropdown menu or F5 will do the trick. A window will come up that you to save the file. Do so. Then another should appear that asks you for some run settings. The first set of three options is under a heading `Console`. I strongly advise selecting the second, which says `Execute in a new dedicated Python console`. Then hit the `Run` button at the bottom and your script will execute.

There's a few things to learn already from your first program. First off, printing is a very important tool for diagnosing problems as it's the easiest way to get information out of a program. Secondly, you've just used a function and a string literal. Here, we'd say you called the function `print` with the argument `'Hello world'`. For now, it suffices to say that functions are things that execute other code when a `()` is placed after them, possibly with other things in the parentheses. Strings are a sequence of characters, and must be delineated with `'` or `"`. For example, `print(Hello world)` will not run, because the names `Hello` and `world` are not defined.

Best practices

Python has two official resources on best practices: PEP 8 and PEP 20. PEP 8 is the official Python language style guide, and contains some fairly strict specifications. The only part of PEP 8 that I disgree with is the 79 character line limit. I'd put a hard limit at 120 characters and say that when you get to 79 characters start thinking about if you're stuffing too much into one line. PEP 20 is the Zen of Python and is much more

the definitive guide on what it means to be pythonic. Anywhere there is a style question that PEP 8 does not explicitly cover, PEP 20 almost certainly will. If PEP 20 doesn't cover it, it doesn't matter.