Python Basics

March 22, 2018

Contents

Introduction **4**

Before Numpy **4**

The Environment and Choices . . . 4
Launching the Jupyter Notebook . . . 4
Launching the IPython QT Console with the Canopy Editor . . . 6
Launching the IPython QT Console From the Terminal . . . 8
Launching the Native Python Console From the Terminal . . . 9
Ending Your Session . . . 9
Data Types and Simple Calculations . . . 9
Hello World . . . 9
`float`, `complex`, `long`, `int`, `str`, and `boolean` . . . 10
Data Structures . . . 17
Lists . . . 17
Tuples . . . 19
Dictionaries . . . 20
Variables . . . 21
Formatting Strings and Gathering User Input . . . 21
Formatting Strings and Printing . . . 21
Gathering User Input . . . 23
Flow Control . . . 25
If, elif, and else . . . 26
For Loops . . . 27
While Loops . . . 28
The Statements `break` and `continue` . . . 29
Exceptions: `try`, `except`, and `finally` Blocks . . . 30
Functions . . . 30
Object Oriented Python: Writing a Class . . . 31
Basics . . . 31
Writing a Simple Class . . . 31

After Numpy **33**
NumPy Fundamentals . . . 33
The N-Dimensional Array and Available Types . . . 33
Array Creation . . . 35
Working With Arrays . . . 36

Graphics and More with Matplotlib **40**

Signals and Systems Tools and Examples **40**
The Scipy Module `scipy.signal` . . . 40
Using `scikit-dsp-comm` . . . 41
More Modules . . . 42
A Simple DSP Class Case Study . . . 42
The `class` Code Base . . . 42
Lowpass and Bandpass Examples . . . 47

References **49**

```
In [1]: %pylab inline
        #%matplotlib qt
        #from __future__ import division # use so 1/2 = 0.5, etc.
        import sk_dsp_comm.sigsys as ss
        import scipy.signal as signal
        from IPython.display import Image, SVG
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: %config InlineBackend.figure_formats=['svg'] # SVG inline viewing
        #%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

```
In [34]: print('Hello World')

Hello World


In [35]: 2*pi

Out[35]: 6.283185307179586

In [36]: arange(0,1,.1)

Out[36]: array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
```

Introduction

This tutorial is structured around the idea that you want to get up and running with Python using `PyLab` as quickly as possible. The first question I asked my myself before I started using PyLab was why consider Python? What makes it a vialble alternative to other languages available for scientific and engineering computations and simulations? OK, everyone has favorites, and presently MATLAB is very popular in the signals and system community. Is there a need to change? This is a debate that lies outside the scope of this tutorial, but the ability to use open-source tools that work really, really well is very compelling.

To answer the first question, why consider Python, I can say:

1. The *NumPy* library
2. combined with *Matplotlib*

3. The *SciPy* library of modules, particularly *signal*, provides reasonable suppost for signals and systems work. Additional libraries of modules are also available

Before Numpy

I have been saying a lot about using Python with Numpy as a means to do scientific and engineering analysis, simulation, and visualization. The fact of the matter is, Python is a good language for doing many other things outside the computational realm.

Numpy plus Scipy are key elements to the attractiveness of using Python, but before getting too carried away with the great scientific computing abiliies of the language, you should learn some basics of the language. This way you will feel more comfortable at coding and debugging.

Before exploring the core language, I will spend time going over the environment and various choices.

The Environment and Choices

How you choose to work with Python is up to you. I do have some strong suggestions. But first I want to review four options in order of most recommended to least recommended. My recommendations assume you are just starting out with Python, so I have a bias towards the Jupyter notebook.

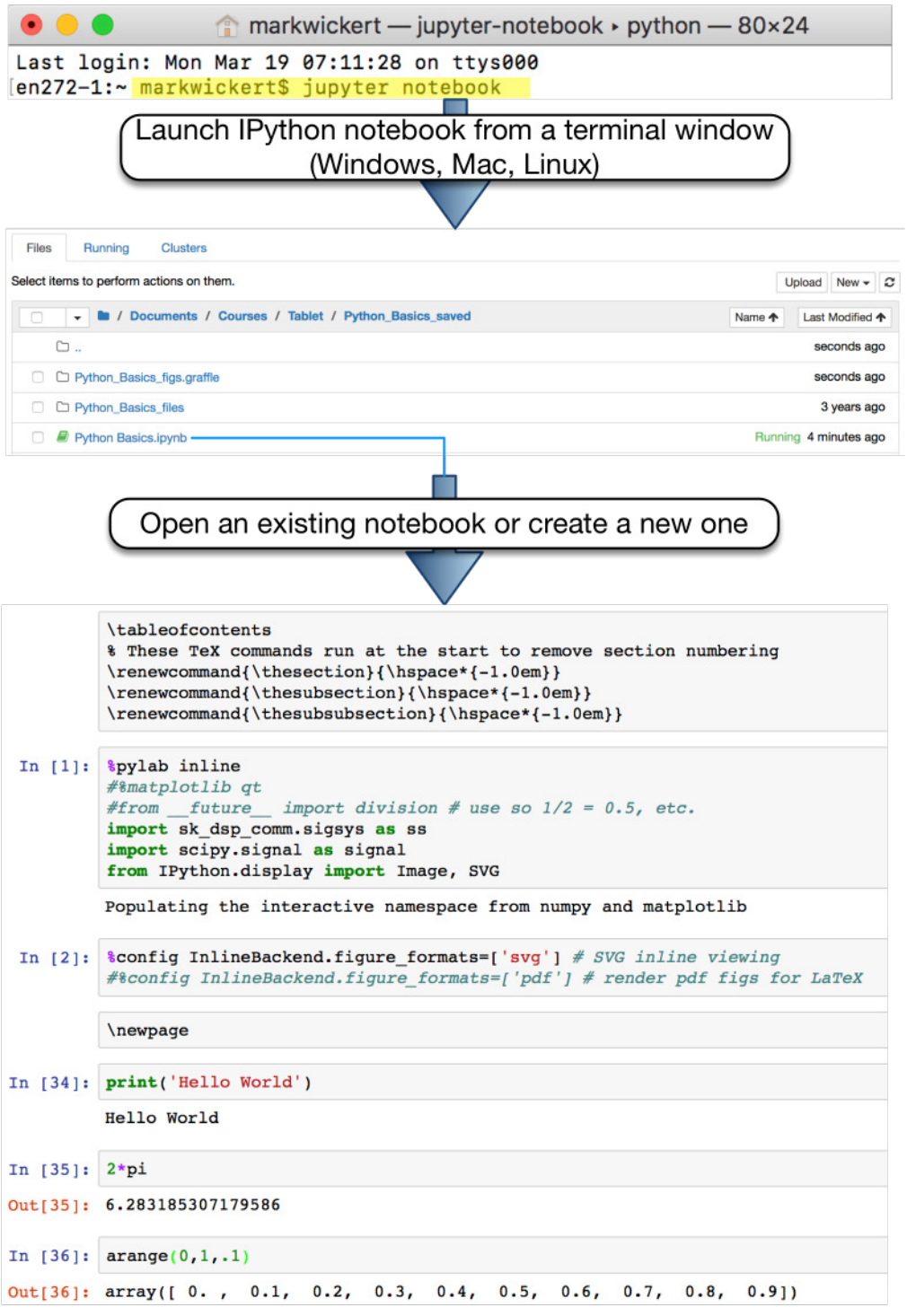
The first thing you want to do is get a version of Python with scientific support included. When this notebook was first created I was using Canopy, but now my preference is to use Anaconda. To learn more about the Jupyter notebook and its furture see Jupyter.

Launching the Jupyter Notebook

Regardless of the operating system, Windows, Mac OS, or Linux, you want to get a terminal window open. It is best if the terminal window is opened at the top level of your user account, so you will be able to navigate to any folder of interest. **Note:** In Windows 10x I recoimment the use of `powershell`. This is done by clicking the *file* menu from the file manager and then selecting *powershell*. It turns out with the notebook interface you can easily navigate to a location interest and then launch an existing notebook or create a new notebook.

```
In [39]: Image('Python_Basics_files/LaunchNotebook2.png',width='90%')
```

Out[39]:



From the above you can see that the notebook is all set. Note that the first cell is only relevant if you intend to render your notebook to pdf using the LaTeX backend. This requires that you