The Python Interpreter

Python is an interpreted language

python Python 3.6.0 | packaged by conda-forge | (default, Jan 13 2017, 23:17:12) [GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux Type "help", "copyright", "credits" or "license" for more information.

In [31]:
```
#hello_world.py
print('Hello world')
```

```
Hello world
```

You can run it by executing the following command:

python hello_world.py

In [32]:
```
import numpy as np
data = {i : np.random.randn() for i in range(7)}
```

In [33]:
```
data
```

Out[33]:
```
{0: 0.9596698244397115,
 1: 0.06612274863669186,
 2: 0.22740627015961948,
 3: -0.9281092977005577,
 4: -1.1535793649745119,
 5: -0.821111375287694,
 6: -0.37855761144907296}
```

Pretty Print

In [34]:
```
print(data)
```

```
{0: 0.9596698244397115, 1: 0.06612274863669186, 2: 0.22740627015961948, 3: -0.9
281092977005577, 4: -1.1535793649745119, 5: -0.821111375287694, 6: -0.378557611
44907296}
```

Introspection

object introspection

Using a question mark (?) before or after a variable will display some general information about the object

In [35]:
```
b = [1, 2, 3]

b?
```

In [36]:
```
print?
```

Using ? shows us the docstring, Using ?? will also show the function’s source code if possible

In [37]:
```
def add_numbers(a, b):
    """
    Add two numbers together
    Returns
    -------
    the_sum : type of arguments
    """
    return a + b
```

In [38]:
```
add_numbers??
```

In [39]:
```
add_numbers?
```

A number of characters combined with the wildcard (*) will show all names matching the wildcard expression

In [40]:
```
import numpy as np
np.*load*?
```

The %run Command

You can execute a .py file by passing the filename to %run

In [41]:
```
pwd
```

Out[41]:
```
'C:\\Users\\User\\1 - Basic Intro'
```

In [42]:
```
%run ipython_script_test.py
```

```
1.4666666666666666
```

All of the variables (imports, functions, and globals) defined in the file (up until an exception, if any, is raised) will then be accessible in the IPython shell:

```
In [43]: c
```

```
Out[43]: 7.5
```

You may also use the related %load magic function, which imports a script into a code cell

```
In [44]: # %load ipython_script_test.py
def f(x, y, z):
    return (x + y) / z


a = 5
b = 6
c = 7.5
result = f(a, b, c)
print(result)
```

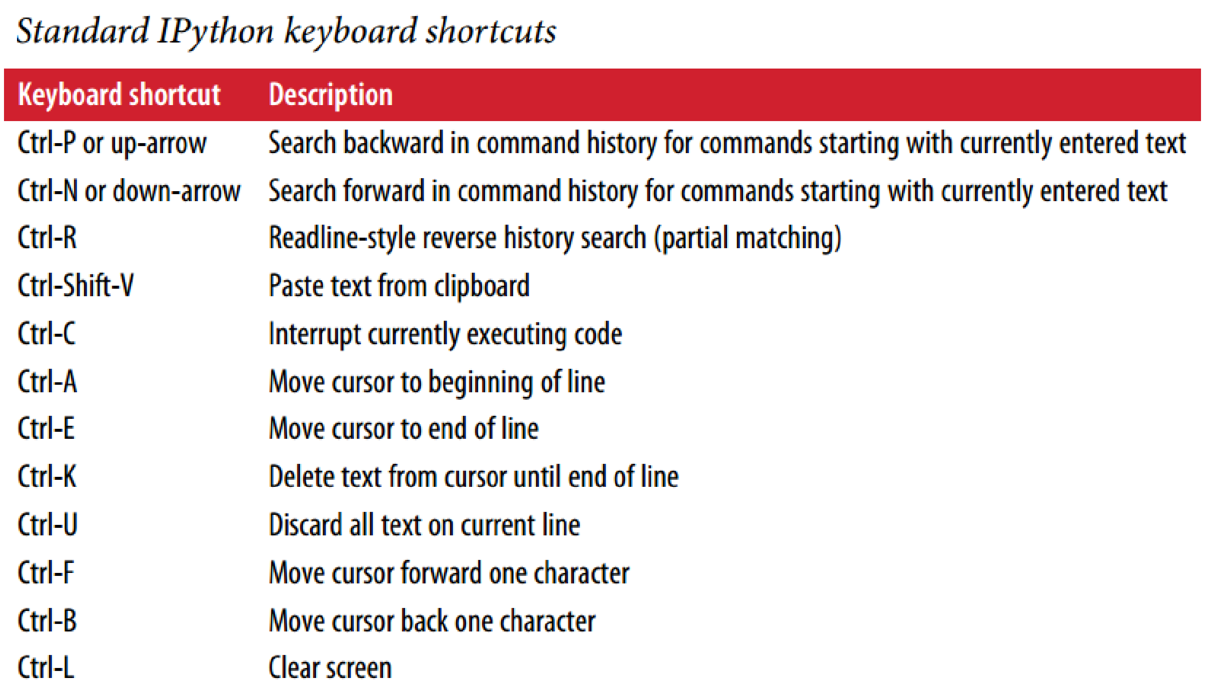
```
1.4666666666666666
```

```
In [45]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

For Ipython

%paste takes whatever text is in the clipboard and executes it as a single block in the shell

%cpaste is similar, except that it gives you a special prompt for pasting code into



Standard IPython keyboard shortcuts

Keyboard shortcut	Description
Ctrl-P or up-arrow	Search backward in command history for commands starting with currently entered text
Ctrl-N or down-arrow	Search forward in command history for commands starting with currently entered text
Ctrl-R	Readline-style reverse history search (partial matching)
Ctrl-Shift-V	Paste text from clipboard
Ctrl-C	Interrupt currently executing code
Ctrl-A	Move cursor to beginning of line
Ctrl-E	Move cursor to end of line
Ctrl-K	Delete text from cursor until end of line
Ctrl-U	Discard all text on current line
Ctrl-F	Move cursor forward one character
Ctrl-B	Move cursor back one character
Ctrl-L	Clear screen

Magic Commands

A magic command is any command prefixed by the percent symbol %.

For example, you can check the execution time of any Python statement, such as a matrix multiplication, using the **%timeit** magic function:

```
In [46]: a = np.random.randn(100, 100)
         %timeit np.dot(a, a)
```

```
48.1 µs ± 5.27 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Many of them have additional “command-line” options, which can all be viewed (as you might expect) using ?:

```
In [47]: %debug?
```

Magic functions can be used by default without the percent sign, as long as no variable is defined with the same name as the magic function in question. This feature is called automagic and can be enabled or disabled with **%automagic**.

Some magic functions behave like Python functions and their output can be assigned to a variable:

```
In [48]: foo = %pwd
```

```
In [49]: foo
```

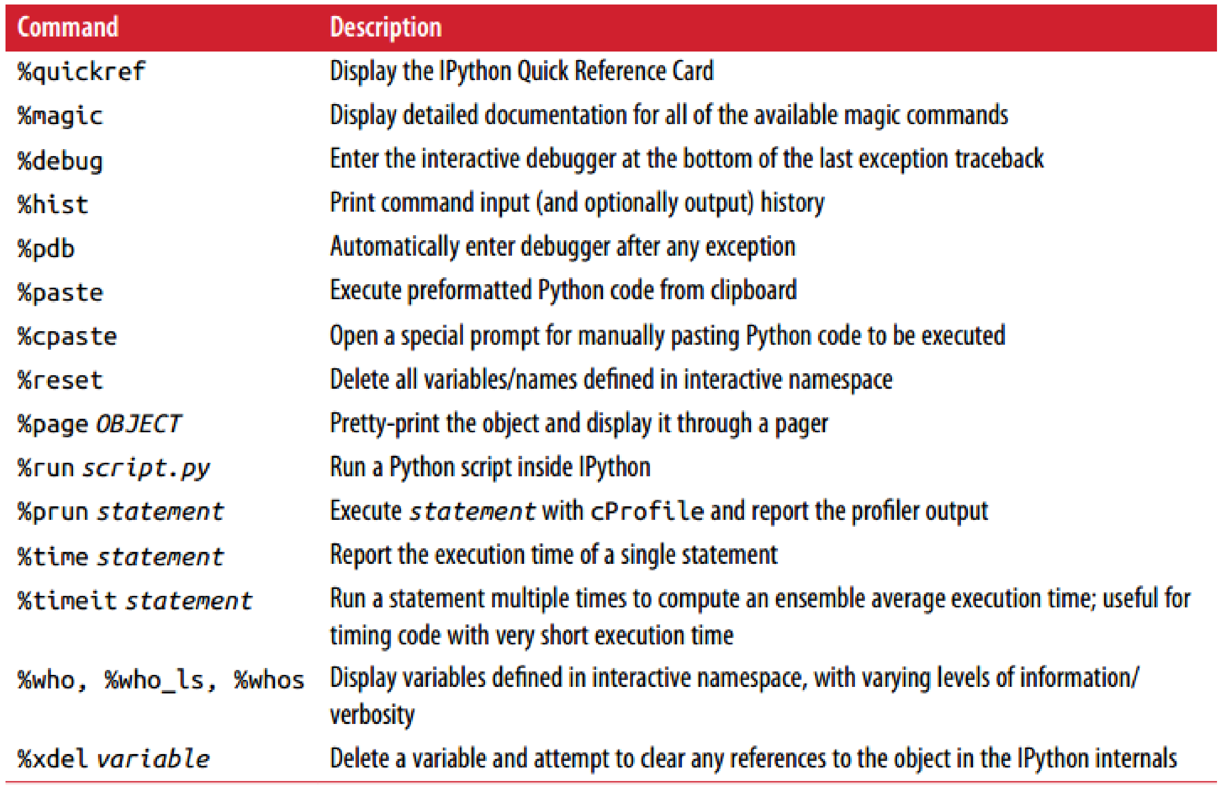
```
Out[49]: 'C:\\Users\\User\\1 - Basic Intro'
```

Explore all of the special commands available by typing **%quickref** or **%magic**:

```
In [50]: %quickref
```

```
In [51]: %magic
```

Some frequently used IPython magic commands



Command	Description
`%quickref`	Display the IPython Quick Reference Card
`%magic`	Display detailed documentation for all of the available magic commands
`%debug`	Enter the interactive debugger at the bottom of the last exception traceback
`%hist`	Print command input (and optionally output) history
`%pdb`	Automatically enter debugger after any exception
`%paste`	Execute preformatted Python code from clipboard
`%cpaste`	Open a special prompt for manually pasting Python code to be executed
`%reset`	Delete all variables/names defined in interactive namespace
`%page` *`OBJECT`*	Pretty-print the object and display it through a pager
`%run` *`script.py`*	Run a Python script inside IPython
`%prun` *`statement`*	Execute *`statement`* with `cProfile` and report the profiler output
`%time` *`statement`*	Report the execution time of a single statement
`%timeit` *`statement`*	Run a statement multiple times to compute an ensemble average execution time; useful for timing code with very short execution time
`%who, %who_ls, %whos`	Display variables defined in interactive namespace, with varying levels of information/verbosity
`%xdel` *`variable`*	Delete a variable and attempt to clear any references to the object in the IPython internals

Matplotlib Integration

The **%matplotlib** magic function configures its integration with the IPython shell or Jupyter notebook. This is important, as otherwise plots you create will either not appear (notebook) or take control of the session until closed (shell).

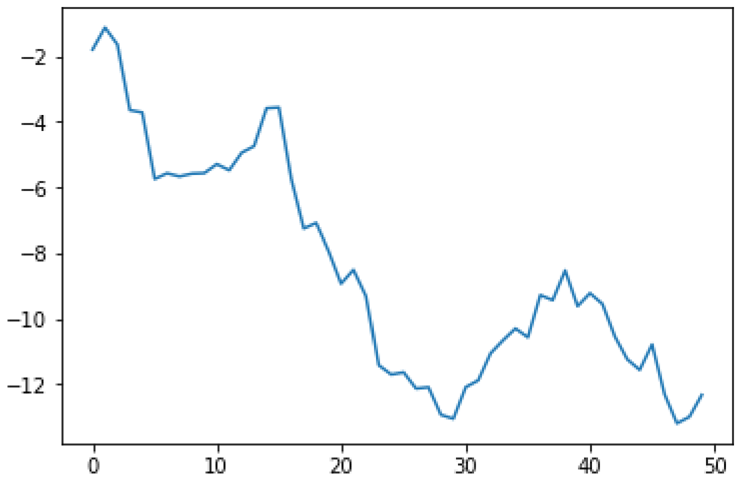
In the IPython shell, running %matplotlib sets up the integration so you can create multiple plot windows without interfering with the console session: **%matplotlib**

In Jupyter, the command is a little different: **%matplotlib inline**

In [53]:
```
%matplotlib inline
```

In [54]:
```
import matplotlib.pyplot as plt
plt.plot(np.random.randn(50).cumsum())
```

Out[54]:
```
[<matplotlib.lines.Line2D at 0xd0f16a0>]
```



In [ ]: