



Python Quick Start Guide

Table of Contents

Python/Environment Setup	2
Text Editor	2
Python Distribution	2
Verifying Your Installation	2
Python Virtual Environments	4
Introduction to Python	6
Create a Work Directory	6
Hello World.....	6
Hello Input	7
Mathematical Expressions	10
Tuples, Lists, and Dictionaries	10
Summary	11
Using Pandas/NumPy Package – Stock Returns Example	12
Running the Python Interpreter	12
Importing Packages	12
Reading In Data From CSV	13
DataFrames.....	14
Viewing Data From DataFrames.....	14
Manipulating Data In DataFrames.....	16
Summary	19

Python/Environment Setup

Note: this tutorial is for individual users working with Python on their own computers. For mass Python deployment, please contact your local IT administration. Python is installed on all the Finance Lab workstations.

Text Editor

A text editor like Notepad, [Notepad++](#), [Notepad2](#), [Atom](#), etc. is required to write and save the Python code presented in the tutorial into .py files. Notepad is already available on all versions of Windows. The recommended Anaconda/Miniconda installers (next section) include the option to install VSCode, another text editor from Microsoft.

Python Distribution

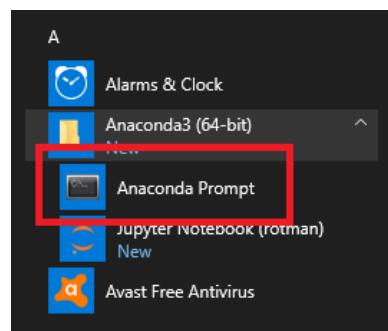
The recommended way to set up your Python environment/workspace is to use either the [Anaconda distribution](#) or the [Miniconda distribution](#) of Python 3.6+

Anaconda already includes many of the most commonly used data science packages (essentially additional tools) like NumPy (support for multidimensional arrays) and Pandas (easy to use data structures and tools for data analysis), as well as a package and virtual environment manager. Miniconda only contains the package and virtual environment manager, and users can manually decide on which packages to download and install for use.

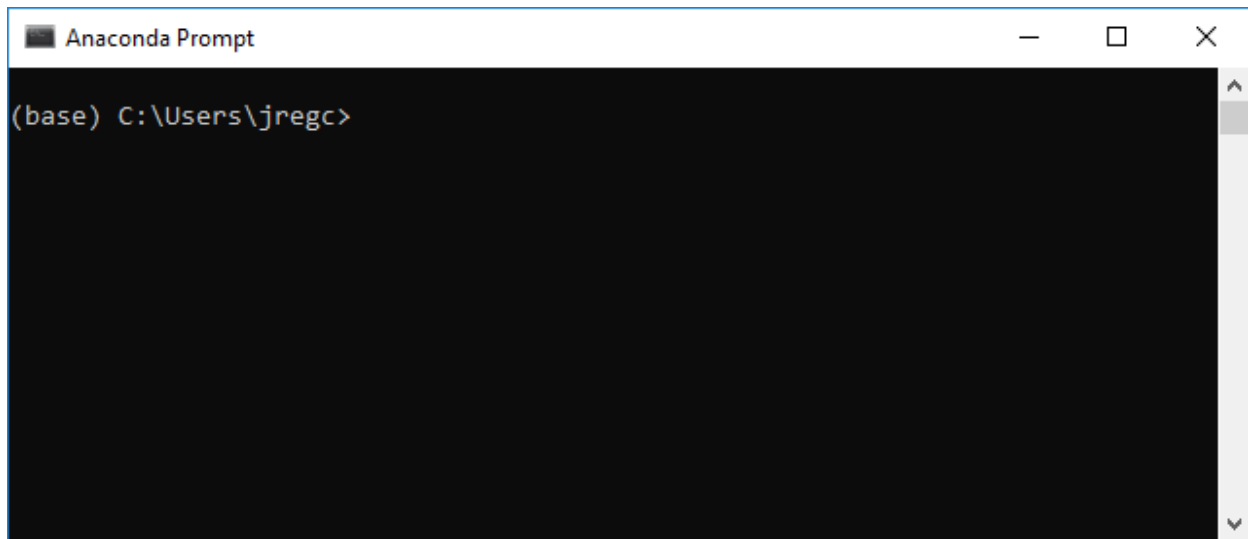
*Note: when installing Anaconda or Miniconda, choose to leave the option 'Add Anaconda to my PATH variable' **unchecked**, and the option 'Register Anaconda as my default Python 3.x' **checked***

Verifying Your Installation

After installing Anaconda or Miniconda, please open the 'Anaconda Prompt' from the Start Menu, or the Command Prompt/PowerShell if you are using a different Python distribution.

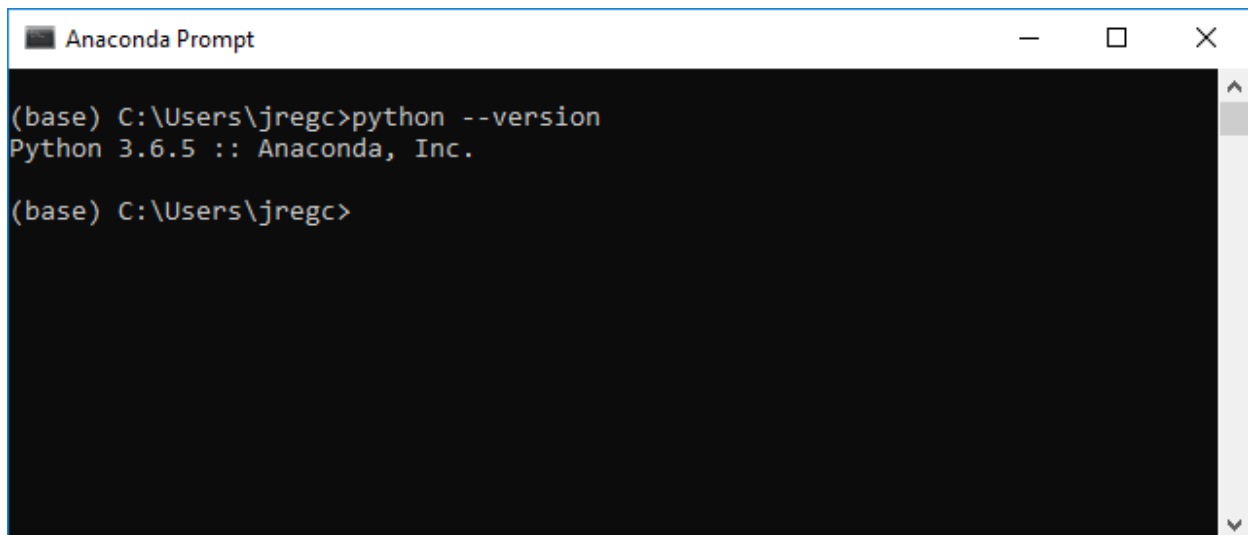


This should open a window looking similar to the following, with 'jregc' being replaced by your user ID. This tutorial will refer to this window as 'the prompt' from here onwards.

A screenshot of an Anaconda Prompt window. The title bar says "Anaconda Prompt". The terminal area is black with white text. The prompt is "(base) C:\Users\jregc>".

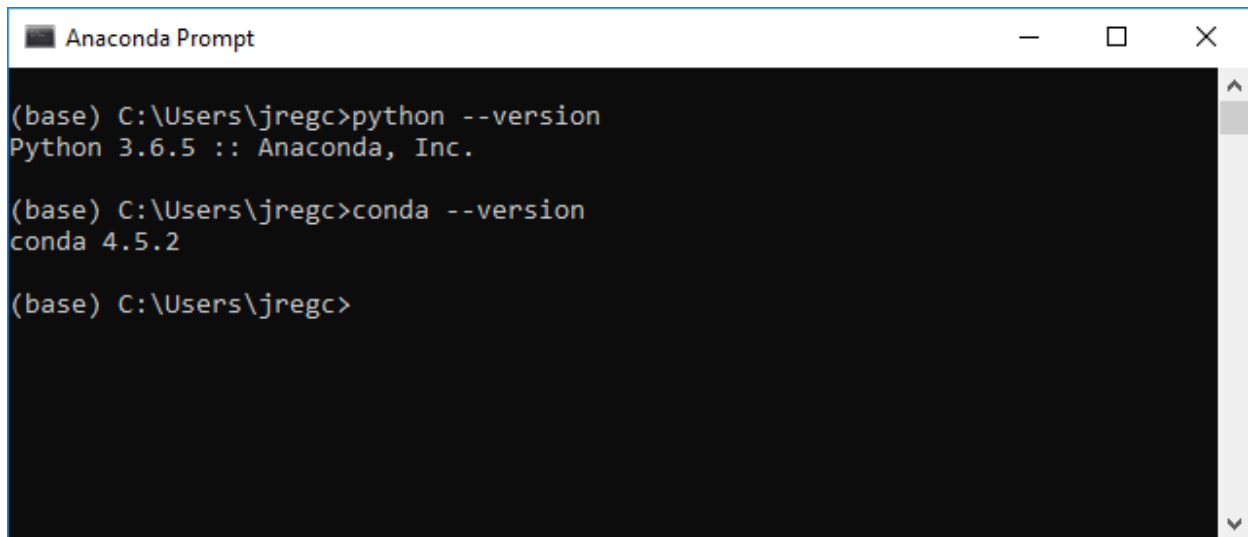
```
(base) C:\Users\jregc>
```

Type `python --version` into the prompt and press `enter`. This command asks Python for its current version number. The output should look similar to the following if everything has been installed correctly, where the version number is 3.6 or greater.

A screenshot of an Anaconda Prompt window. The title bar says "Anaconda Prompt". The terminal area is black with white text. The prompt is "(base) C:\Users\jregc>". The user has entered the command "python --version" and the output is "Python 3.6.5 :: Anaconda, Inc.". The prompt is now "(base) C:\Users\jregc>".

```
(base) C:\Users\jregc>python --version
Python 3.6.5 :: Anaconda, Inc.
(base) C:\Users\jregc>
```

Then type `conda --version` into the prompt and press `enter`. This command asks Anaconda/Miniconda for its current version number. The output should look similar to the following if everything has been installed correctly, where the version number is 4.5 or greater. In the case where the version number is lower than 4.5, type `conda update -n base conda` to get the latest version.



```

Anaconda Prompt

(base) C:\Users\jregc>python --version
Python 3.6.5 :: Anaconda, Inc.

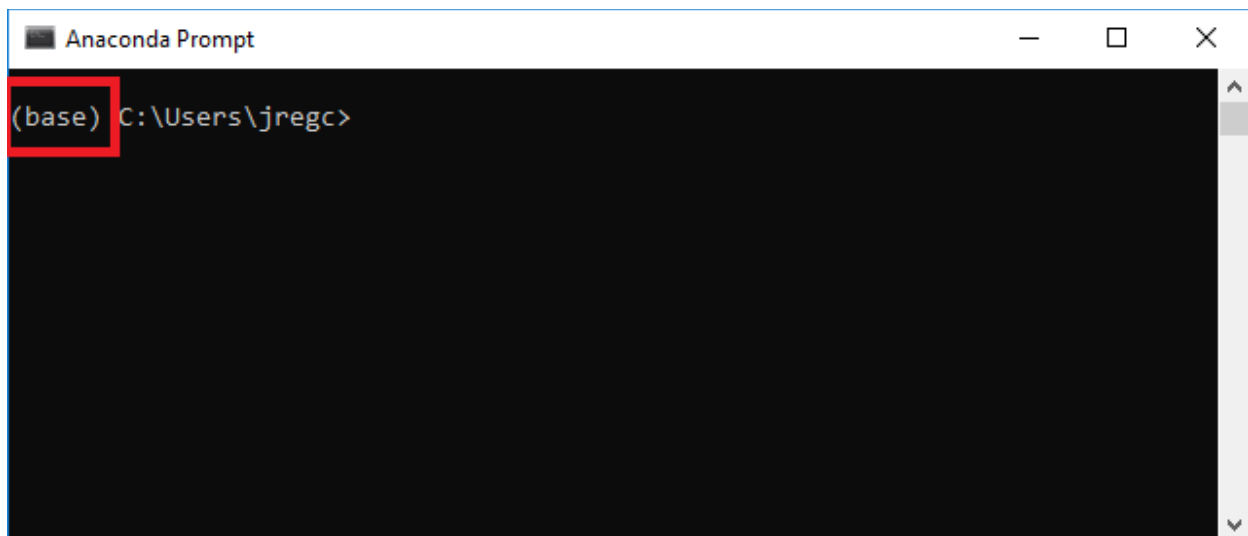
(base) C:\Users\jregc>conda --version
conda 4.5.2

(base) C:\Users\jregc>
```

Python Virtual Environments

Anaconda and Miniconda come with the `conda` package and virtual environment manager. Different Python applications that users write may require different files and packages, and virtual environments help solve this problem. A virtual environment is a self-contained environment/directory that contains its own files, installed packages, and their dependencies that will not interact with other environments' files, packages, and dependencies.

When a user initially starts the prompt, it starts in the 'base' environment, as indicated on the left side of the prompt.

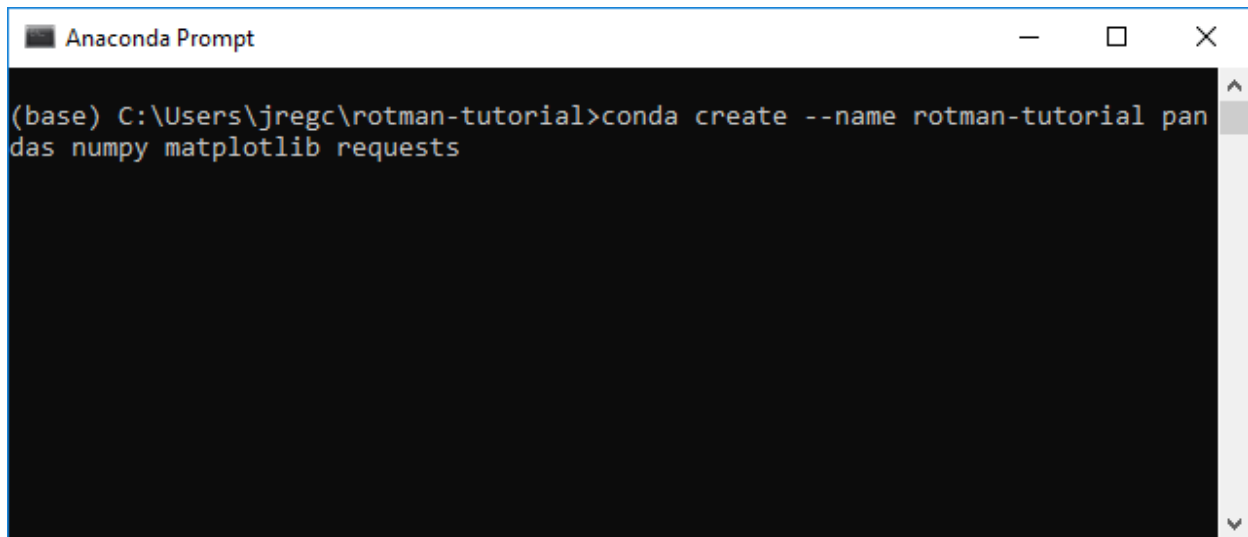


```

Anaconda Prompt

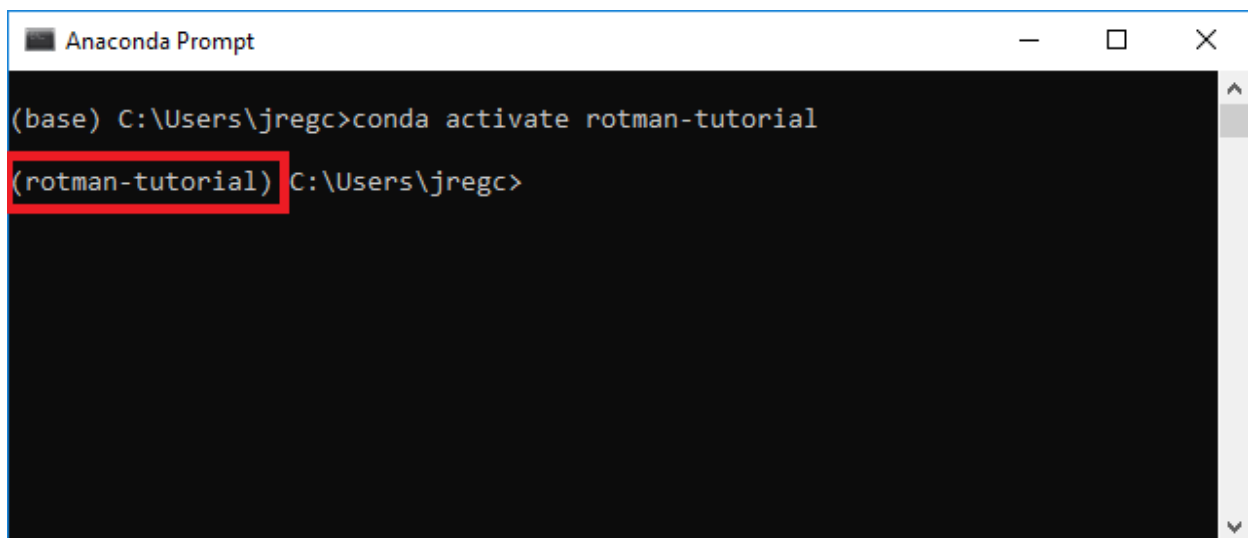
(base) C:\Users\jregc>
```

However, it is not recommended to install additional packages in the 'base' environment. To create a new environment, enter `conda create --name <ENV NAME> pandas numpy matplotlib requests`. This will create a new virtual environment, with the name supplied in `<ENV NAME>`, and with the 'pandas', 'numpy', 'matplotlib', and 'requests' packages needed in this tutorial, plus any dependencies for those packages.



```
Anaconda Prompt
(base) C:\Users\jregc\rotman-tutorial>conda create --name rotman-tutorial pandas numpy matplotlib requests
```

In this case, the virtual environment is named 'rotman-tutorial'. Enter `y` into the prompt after `conda` lists the packages that must be downloaded and installed to proceed and create the environment. After the environment is created, enter `conda activate <ENV NAME>` or simply `activate <ENV NAME>` into the prompt to switch the context of the prompt to that environment.



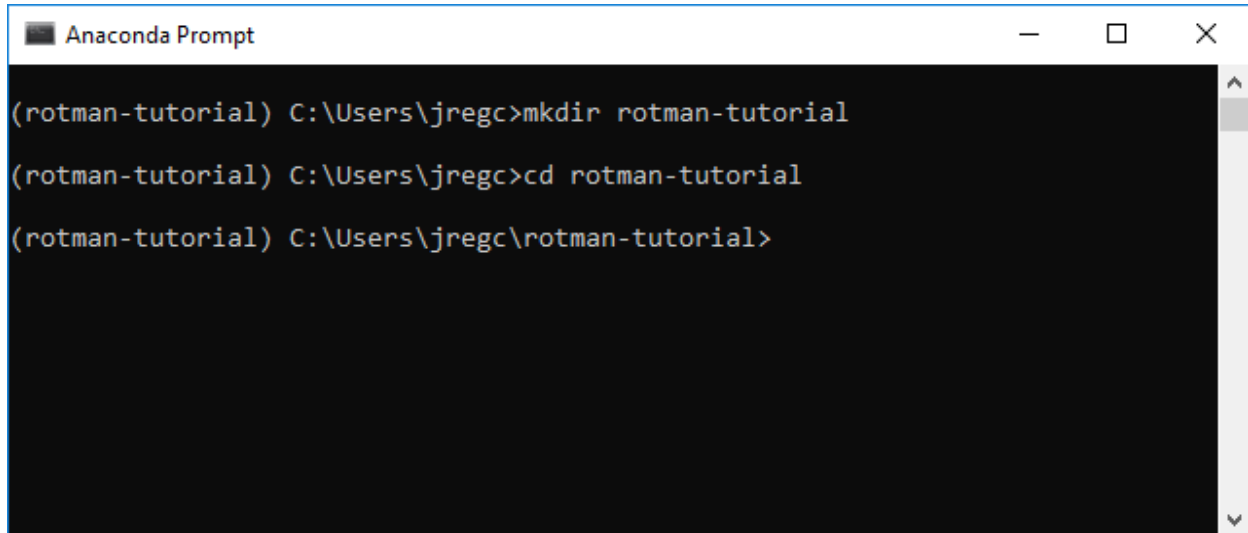
```
Anaconda Prompt
(base) C:\Users\jregc>conda activate rotman-tutorial
(rotman-tutorial) C:\Users\jregc>
```

As shown in the above screenshot, after entering `conda activate rotman-tutorial` into the prompt, the prompt indicates that the current environment is 'rotman-tutorial'. If a user wants to deactivate the current environment and go back to the 'base' environment, enter `conda deactivate`.

Introduction to Python

Create a Work Directory

In the local user directory, create a work directory to store the tutorial files. Users can do this from the prompt by entering `mkdir <WORK DIR NAME>` to create a directory in the current location. Then, enter `cd <PATH TO WORK DIR>` to change locations to that directory.

A screenshot of an Anaconda Prompt window. The title bar says "Anaconda Prompt". The command prompt shows three lines of text:

```
(rotman-tutorial) C:\Users\jregc>mkdir rotman-tutorial
(rotman-tutorial) C:\Users\jregc>cd rotman-tutorial
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>
```

In the above screenshot, a directory called 'rotman-tutorial' was created in the directory `C:\Users\jregc`

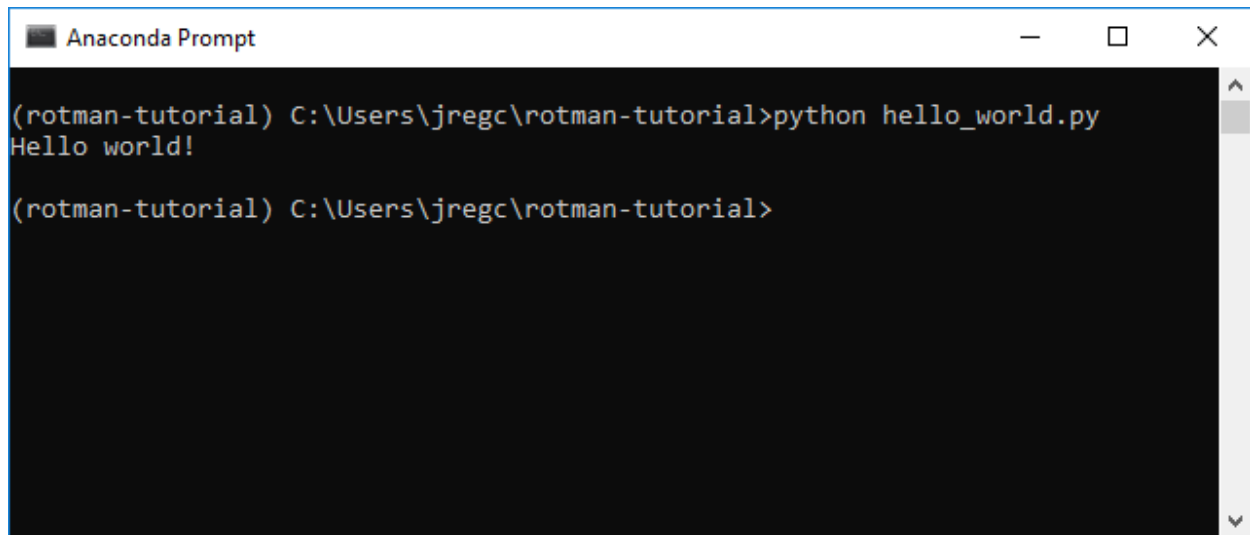
Hello World

Open your preferred text editor, type the following into a new file, and save the file in the work directory as `hello_world.py`.

```
def main():
    print('Hello world!')

# this if-block tells Python to call the main() method when it runs the file
from the prompt
if __name__ == '__main__':
    main()
```

Then in the prompt, enter `python hello_world.py`.

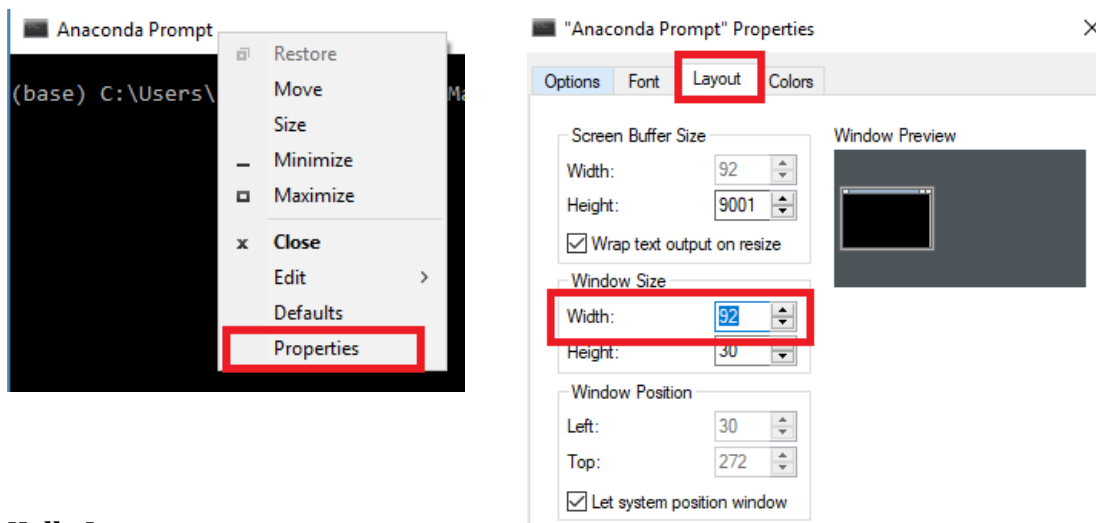


```
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python hello_world.py
Hello world!

(rotman-tutorial) C:\Users\jregc\rotman-tutorial>
```

This command tells Python to run the file in the local directory called `hello_world.py`. Inside that file, there is a method called **main** that calls the **print** method. The **print** method takes in the text 'Hello world' as a parameter and prints it out to the prompt as `Hello world!`.

In case the prompt window size needs to be changed, right-click on the top module bar from the prompt window, choose "Properties", click on "Layout", and change "Width" under "Window Size" to display any contents properly.



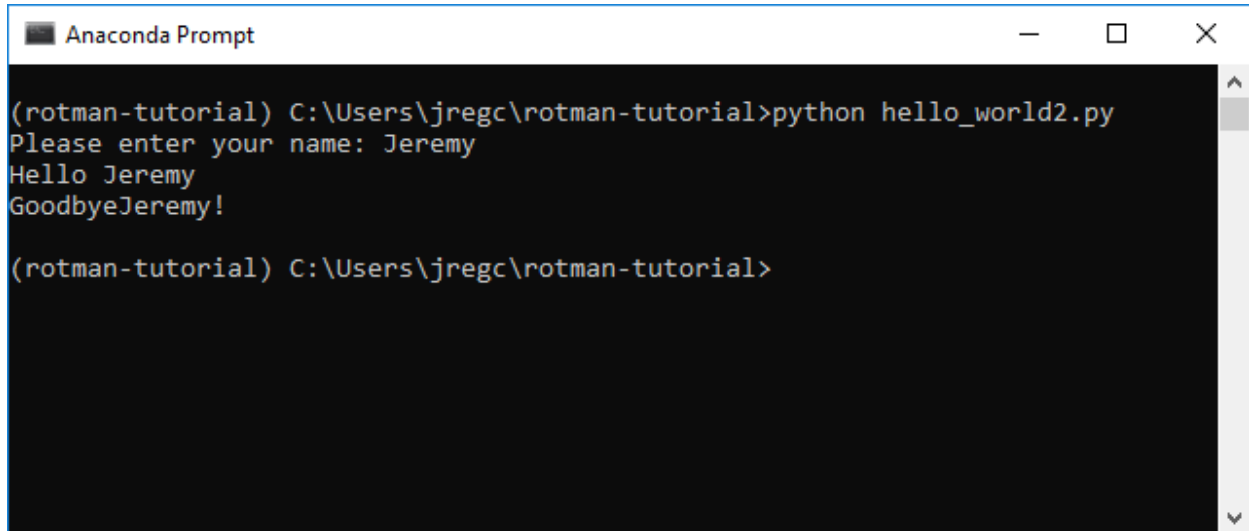
Hello Input

Python can also take in user input. For example, try saving the following into a file called `hello_world2.py` and running it:

```
def main():
    name = input('Please enter your name: ')
    print('Hello', name)
    print('Goodbye' + name + '!')

if __name__ == '__main__':
    main()
```

This time, a prompt should be displayed, asking for your name. In effect, the first line of code tells Python to print to the prompt the text `Please enter your name:`, wait for an input to be typed in, and then save that input into the variable called **name**. The second line then tells Python to print `Hello` and the value saved in the variable **name**. The third line shows another way of combining text together to be printed out.



```
Anaconda Prompt

(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python hello_world2.py
Please enter your name: Jeremy
Hello Jeremy
GoodbyeJeremy!

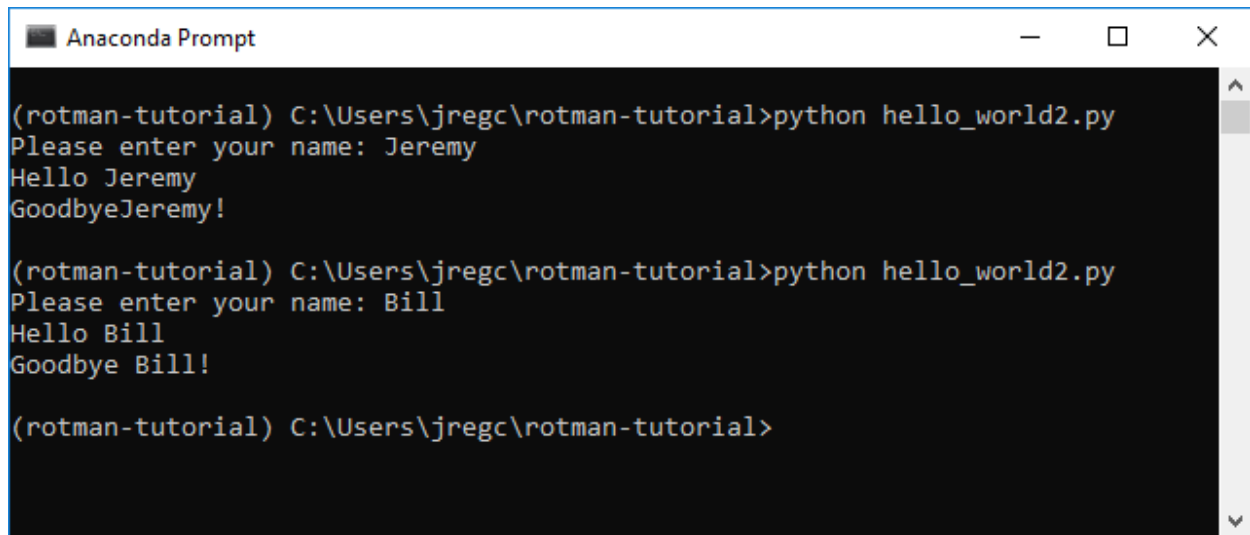
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>
```

But if you look at what's displayed on the third line of the output, it looks a little messy. Let's fix that:

```
def main():
    name = input('Please enter your name: ')
    print('Hello', name)
    print('Goodbye ' + name + '!')

if __name__ == '__main__':
    main()
```

Note the space in the quoted text 'Goodbye '.

A screenshot of an Anaconda Prompt window. The title bar reads "Anaconda Prompt" and includes standard window controls (minimize, maximize, close). The terminal content shows the execution of a Python script named "hello_world2.py". The script prompts the user for their name and prints a greeting and goodbye message. The first run uses the name "Jeremy", and the second run uses the name "Bill". The prompt character is a greater-than sign (>).

```
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python hello_world2.py
Please enter your name: Jeremy
Hello Jeremy
GoodbyeJeremy!

(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python hello_world2.py
Please enter your name: Bill
Hello Bill
Goodbye Bill!

(rotman-tutorial) C:\Users\jregc\rotman-tutorial>
```

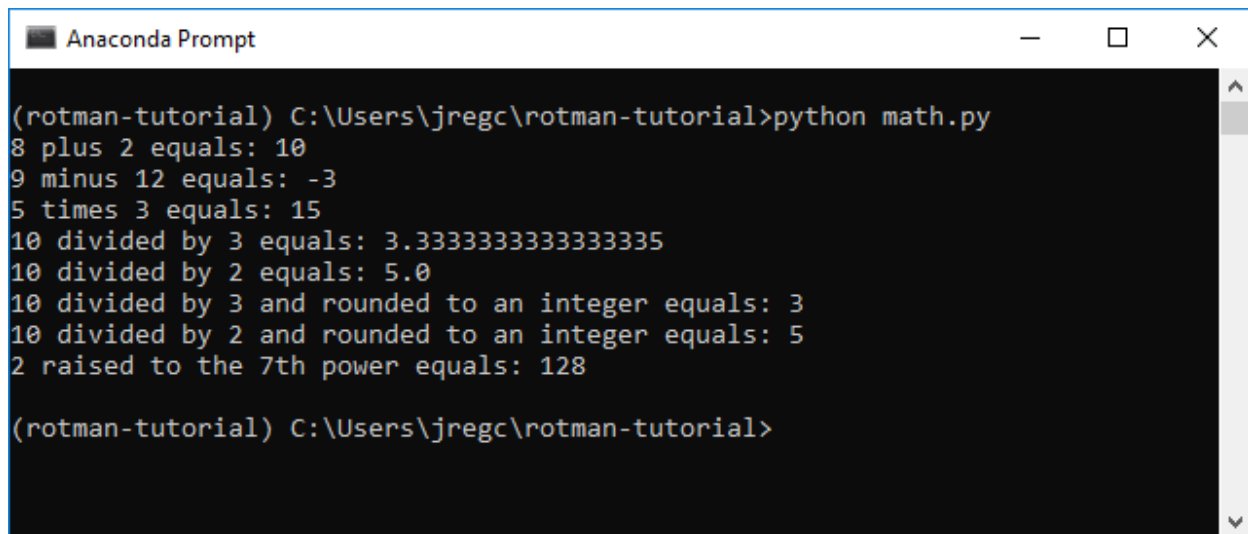
There, that's better!

Mathematical Expressions

Like many programming languages, Python can also perform mathematical calculations. Try saving and running the following as `math.py`:

```
def main():
    print('8 plus 2 equals:', 8 + 2)
    print('9 minus 12 equals:', 9 - 12)
    print('5 times 3 equals:', 5 * 3)
    print('10 divided by 3 equals:', 10 / 3)
    print('10 divided by 2 equals:', 10 / 2)
    print('10 divided by 3 and rounded to an integer equals:', 10 // 3)
    print('10 divided by 2 and rounded to an integer equals:', 10 // 2)
    print('2 raised to the 7th power equals:', 2 ** 7)

if __name__ == '__main__':
    main()
```



```
Anaconda Prompt
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python math.py
8 plus 2 equals: 10
9 minus 12 equals: -3
5 times 3 equals: 15
10 divided by 3 equals: 3.3333333333333335
10 divided by 2 equals: 5.0
10 divided by 3 and rounded to an integer equals: 3
10 divided by 2 and rounded to an integer equals: 5
2 raised to the 7th power equals: 128
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>
```

Note that there is a difference between integer and floating-point math, where floating-point numbers are representations of real numbers including decimals.

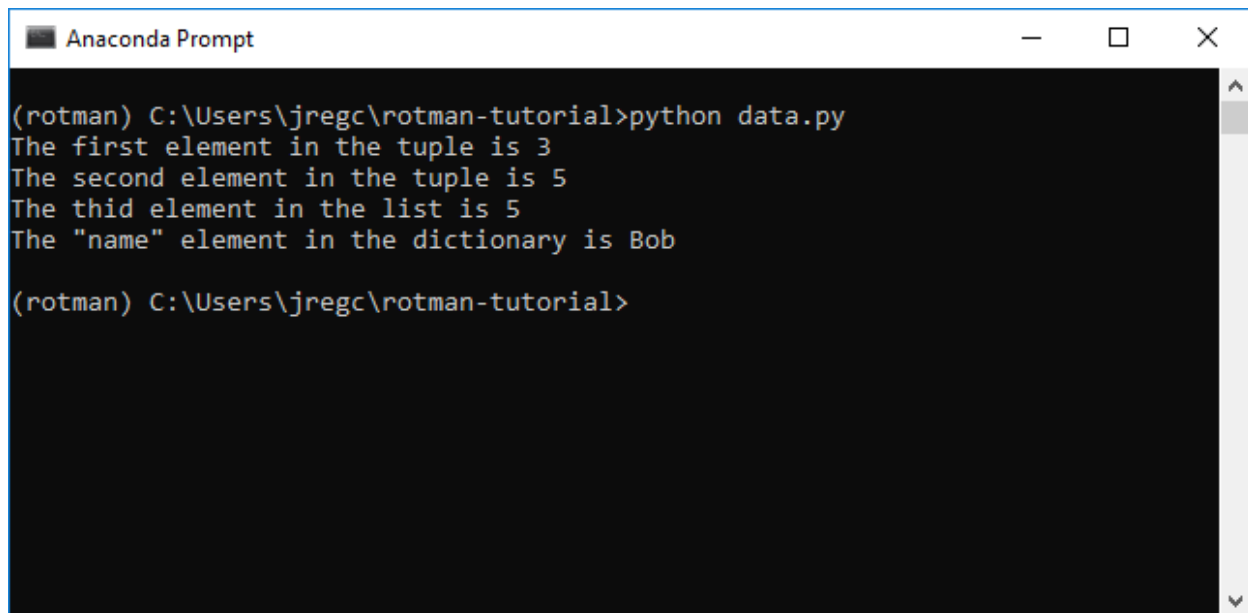
Tuples, Lists, and Dictionaries

There are also three common data structures that are used in Python: tuples, lists, and dictionaries. Tuples are comma-separated lists of values that cannot be changed once created, while lists are comma-separated lists of values that can be changed. Dictionaries are lists of key/value pairs that are associated with one another. In effect, the major difference is how to access values in the different data structures: usually one will index by number to access values in tuples and lists, while one will index by key to access a value in a dictionary. The following example illustrates how this works.

```
def main():
    t = (3, 5, 10, 9)
    l = [8, 9, 5]
    d = {'key': 'value', 'name': 'Bob'}

    print('The first element in the tuple is', t[0])
    print('The second element in the tuple is', t[1])
    print('The third element in the list is', l[2])
    print('The "name" element in the dictionary is', d['name'])

if __name__ == '__main__':
    main()
```



```
Anaconda Prompt
(rotman) C:\Users\jregc\rotman-tutorial>python data.py
The first element in the tuple is 3
The second element in the tuple is 5
The thid element in the list is 5
The "name" element in the dictionary is Bob

(rotman) C:\Users\jregc\rotman-tutorial>
```

Note that python uses 0-based indexing, such that the first element is at position 0, the second is at position 1, etc.

Summary

This concludes a basic introduction to Python, necessary for the following sections on using Pandas/NumPy for simple stock return calculations. You should now be able to write a simple set of instructions (a method) in Python, using a pre-defined method (print) and execute it from the prompt.

For a more detailed introduction to Python, please see [The Python Tutorial](#).

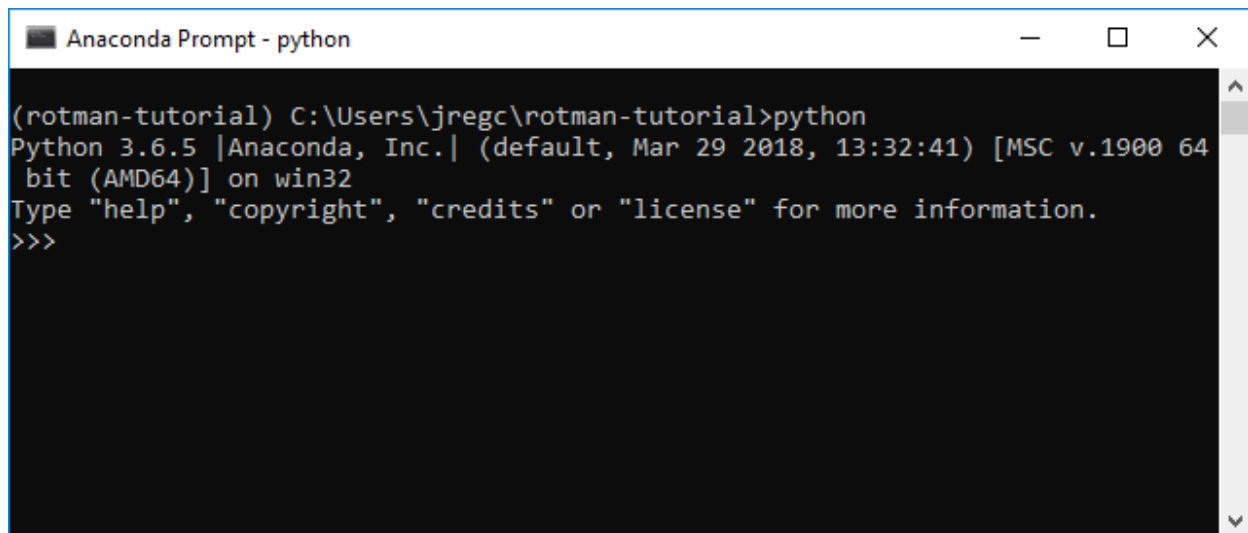
Using Pandas/NumPy Package – Stock Returns Example

Pandas is a commonly used open-source data analysis package for Python. It provides a comprehensive set of easy-to-use data structures and analysis tools. We'll take a quick look at how to use Pandas to read in CSV data from Yahoo Finance and perform some common calculations like returns and summary statistics.

Instead of writing the code into a file and then running it via `python <FILE NAME>.py`, we'll use the interactive Python interpreter available via the prompt. Note however that the code can also be saved into a `.py` file and run, as demonstrated in the [Introduction to Python](#) section.

Running the Python Interpreter

To run the Python interpreter, simply enter `python` into the prompt, first ensuring that the 'rotman-tutorial' (or other) virtual environment is active and the prompt is in your working directory.

A screenshot of the Anaconda Prompt window. The title bar reads "Anaconda Prompt - python". The window has standard Windows window controls (minimize, maximize, close) on the right. The terminal area has a black background with white text. The text shows the command prompt at "C:\Users\jregc\rotman-tutorial>python", followed by the Python version and environment information: "Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32". It then displays the help text: "Type 'help', 'copyright', 'credits' or 'license' for more information." and the interactive prompt ">>>".

```
Anaconda Prompt - python
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

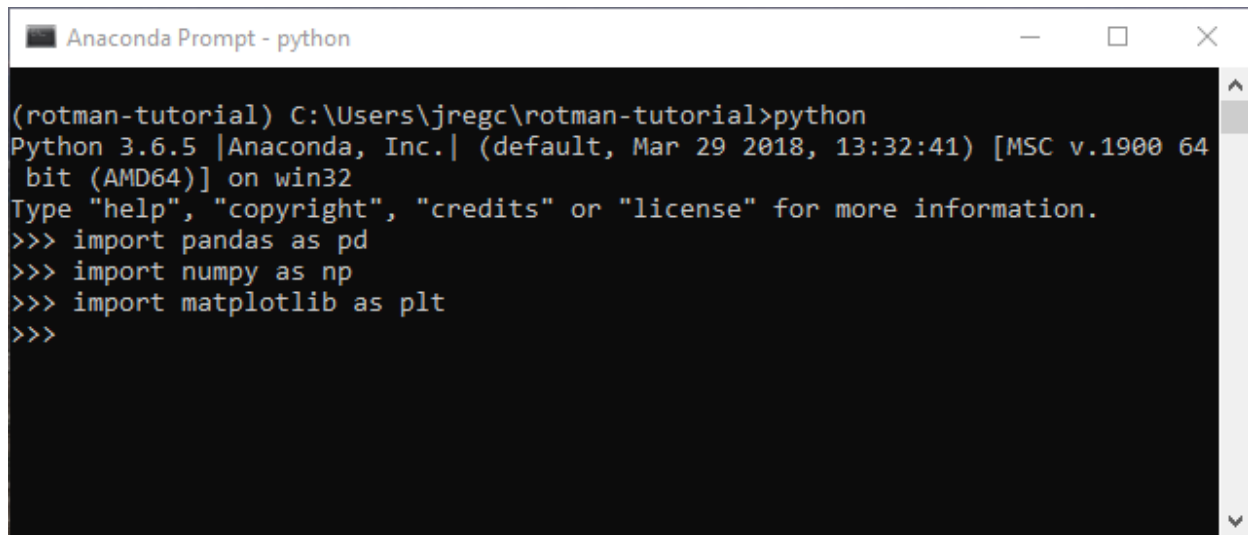
As the screenshot shows, the Python interpreter is active, running Python version 3.6.5. The `>>>` shows that we are in interactive mode, and can enter commands to be interpreted by Python.

To exit the Python interpreter, enter the command `exit()`.

Importing Packages

To import packages, either into a Python file or into the interpreter, type the following lines:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

A screenshot of an Anaconda Prompt window titled "Anaconda Prompt - python". The window shows a Python 3.6.5 shell with the following text: (rotman-tutorial) C:\Users\jregc\rotman-tutorial>python, Python 3.6.5 [Anaconda, Inc.] (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32, Type "help", "copyright", "credits" or "license" for more information., >>> import pandas as pd, >>> import numpy as np, >>> import matplotlib as plt, >>>. The prompt is on a black background with white text.

```
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python
Python 3.6.5 [Anaconda, Inc.] (default, Mar 29 2018, 13:32:41) [MSC v.1900 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib as plt
>>>
```

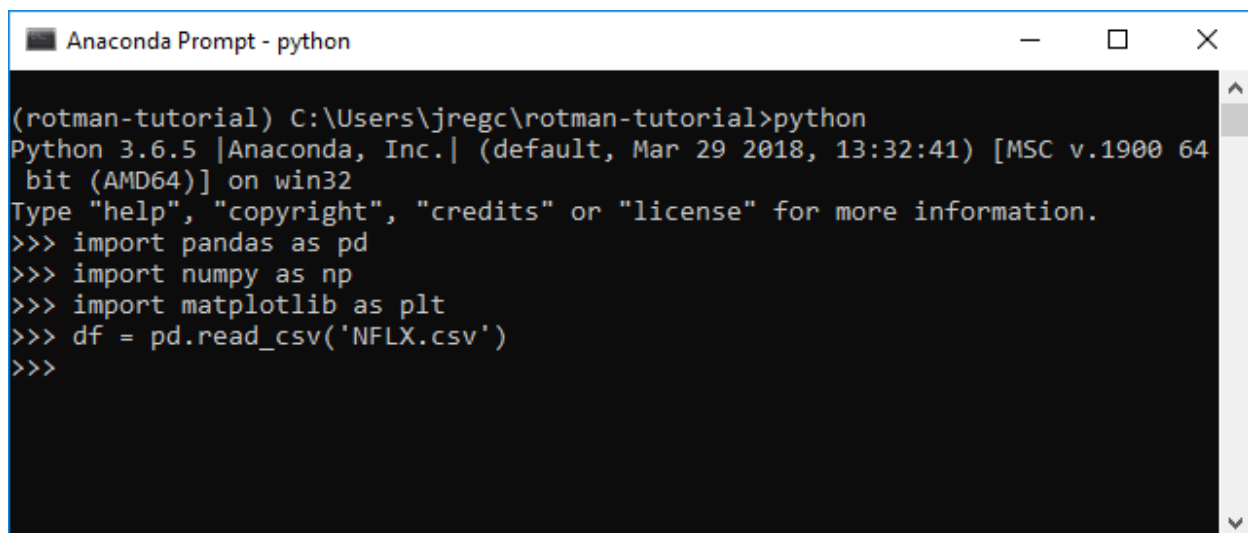
These three lines import the 'pandas', 'numpy', and 'matplotlib.pyplot' packages that we installed in the 'rotman-tutorial' virtual environment which was set up in the [Python Virtual Environments](#) section of the tutorial. Additionally, we create nicknames to reference them by ('pd', 'np', and 'plt' respectively). The next section will show how to call methods from these packages.

Reading In Data From CSV

Let's get some data in the form of a CSV file to read. Go to [Yahoo Finance](#), query an equity ticker, and download a 1Y span of historical daily data. Save this CSV in your work directory, as set up in the [Create A Work Directory](#) section.

In this tutorial, we're using Netflix (NFLX) historical data.

```
df = pd.read_csv('NFLX.csv')
```

A screenshot of an Anaconda Prompt window titled "Anaconda Prompt - python". The window shows a Python 3.6.5 shell with the following text: (rotman-tutorial) C:\Users\jregc\rotman-tutorial>python, Python 3.6.5 [Anaconda, Inc.] (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32, Type "help", "copyright", "credits" or "license" for more information., >>> import pandas as pd, >>> import numpy as np, >>> import matplotlib as plt, >>> df = pd.read_csv('NFLX.csv'), >>>. The prompt is on a black background with white text.

```
(rotman-tutorial) C:\Users\jregc\rotman-tutorial>python
Python 3.6.5 [Anaconda, Inc.] (default, Mar 29 2018, 13:32:41) [MSC v.1900 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib as plt
>>> df = pd.read_csv('NFLX.csv')
>>>
```

This command calls the `read_csv()` method available in the 'pandas' package, passing in the filename 'NFLX.csv' as the parameter specifying the file to open and read in the same directory. Relative paths

are also possible, for example `pd.read_csv('data/NFLX.csv')` would read a 'NFLX.csv' file located in a subdirectory named 'data'.

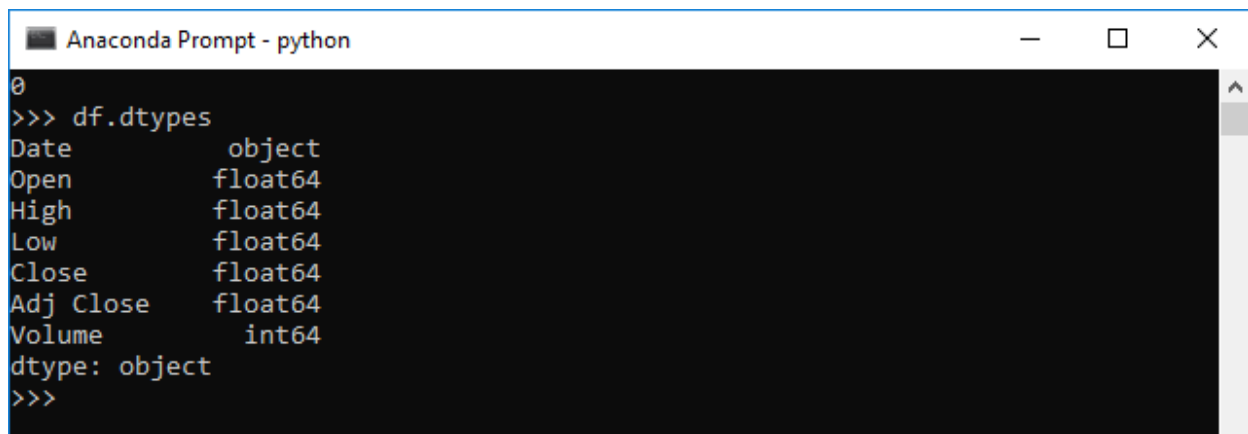
After reading the data in from the CSV file, the `read_csv()` method returns it as a DataFrame object, and the variable named **df** (for DataFrame) refers to that DataFrame object.

DataFrames

DataFrames are the primary data structure in Pandas, and can be thought of as two dimensional tables with labeled axes, similar to how data is laid out in a `.csv` or `.xls/.xlsx` file in rows and columns.

Viewing Data From DataFrames

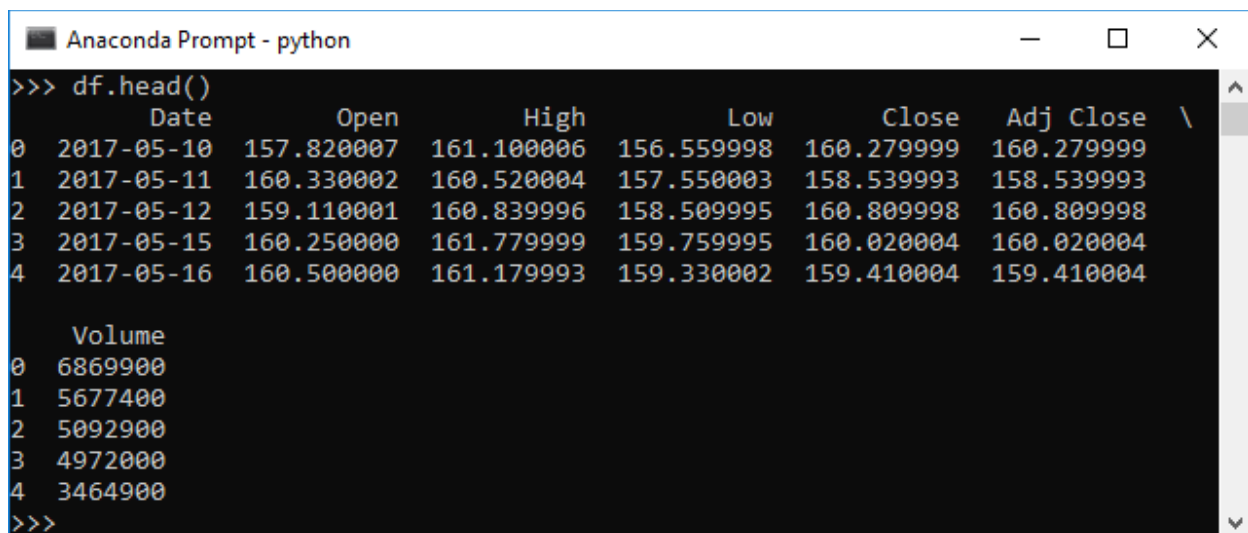
`df.dtypes`



```
Anaconda Prompt - python
>>> df.dtypes
Date          object
Open          float64
High          float64
Low           float64
Close         float64
Adj Close     float64
Volume        int64
dtype: object
>>>
```

The `dtypes` attribute provides a list of the data types of each column.

`df.head()`

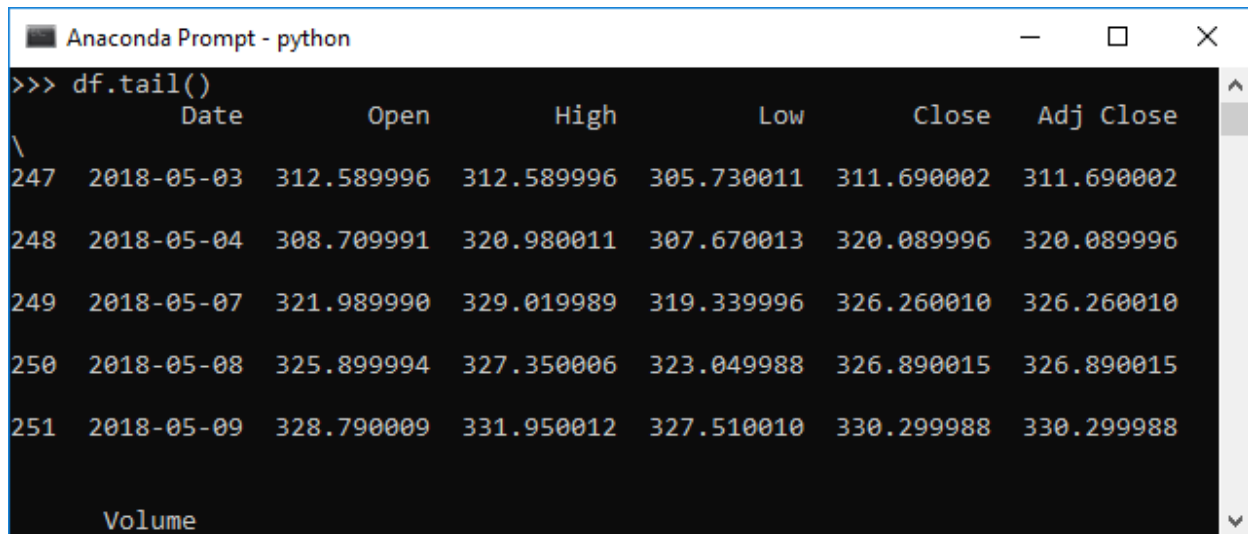


```
Anaconda Prompt - python
>>> df.head()
   Date      Open      High      Low      Close  Adj Close \
0  2017-05-10  157.820007  161.100006  156.559998  160.279999  160.279999
1  2017-05-11  160.330002  160.520004  157.550003  158.539993  158.539993
2  2017-05-12  159.110001  160.839996  158.509995  160.809998  160.809998
3  2017-05-15  160.250000  161.779999  159.759995  160.020004  160.020004
4  2017-05-16  160.500000  161.179993  159.330002  159.410004  159.410004

   Volume
0  6869900
1  5677400
2  5092900
3  4972000
4  3464900
>>>
```

The `head()` method displays the first 5 rows in the DataFrame. A different number of rows to display can be passed in as a parameter (for example `df.head(10)` would display the first 10 rows).

```
df.tail()
```

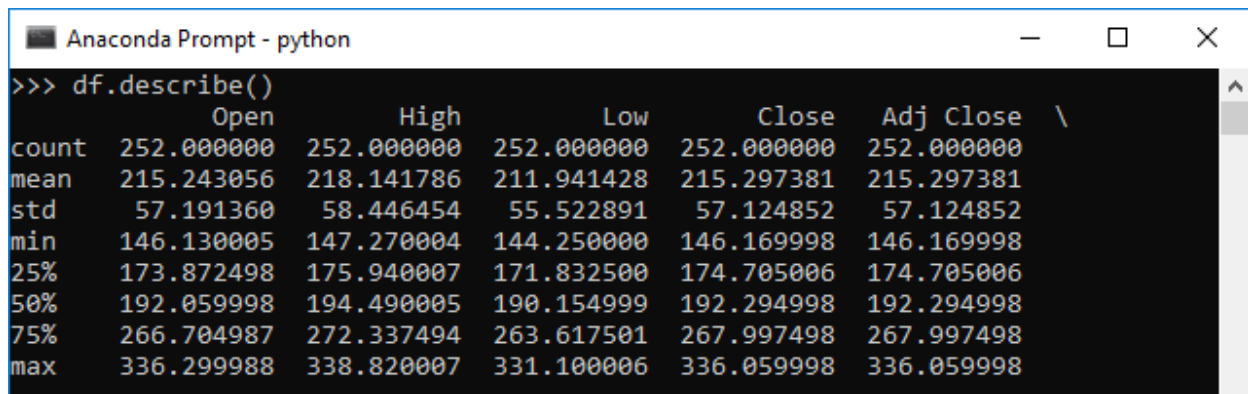


The screenshot shows the output of the `df.tail()` command in an Anaconda Prompt window. The output displays the last 5 rows of a DataFrame, indexed 247 to 251. The columns are Date, Open, High, Low, Close, and Adj Close. The 'Volume' column is partially visible at the bottom.

	Date	Open	High	Low	Close	Adj Close
247	2018-05-03	312.589996	312.589996	305.730011	311.690002	311.690002
248	2018-05-04	308.709991	320.980011	307.670013	320.089996	320.089996
249	2018-05-07	321.989990	329.019989	319.339996	326.260010	326.260010
250	2018-05-08	325.899994	327.350006	323.049988	326.890015	326.890015
251	2018-05-09	328.790009	331.950012	327.510010	330.299988	330.299988

The `tail()` method displays the last 5 rows in the DataFrame. A different number of rows to display can be passed in as a parameter (for example `df.tail(20)` would display the last 20 rows).

```
df.describe()
```



The screenshot shows the output of the `df.describe()` command in an Anaconda Prompt window. The output displays summary statistics for the columns: count, mean, std, min, 25%, 50%, 75%, and max. The columns are Open, High, Low, Close, and Adj Close.

	Open	High	Low	Close	Adj Close
count	252.000000	252.000000	252.000000	252.000000	252.000000
mean	215.243056	218.141786	211.941428	215.297381	215.297381
std	57.191360	58.446454	55.522891	57.124852	57.124852
min	146.130005	147.270004	144.250000	146.169998	146.169998
25%	173.872498	175.940007	171.832500	174.705006	174.705006
50%	192.059998	194.490005	190.154999	192.294998	192.294998
75%	266.704987	272.337494	263.617501	267.997498	267.997498
max	336.299988	338.820007	331.100006	336.059998	336.059998

The `describe()` method calculates and displays some common sample statistics for the DataFrame's columns, including the count, mean, standard deviation, min/max values, and quartiles. It skips NA values.

```
df['Adj Close']
```

```
Anaconda Prompt - python
>>> df['Adj Close']
0      160.279999
1      158.539993
2      160.809998
3      160.020004
4      159.410004
5      153.199997
6      155.699997
7      157.020004
8      157.160004
9      157.949997
10     157.750000
11     163.050003
12     162.429993
13     163.220001
```

Display a column from the DataFrame, selected by label.

Manipulating Data In DataFrames

It's also possible to add new columns to a DataFrame and perform other calculations:

```
df['Daily Return'] = df['Adj Close'].pct_change()
df['10DMA'] = df['Adj Close'].rolling(window=10, center=False).mean()
df['30DMA'] = df['Adj Close'].rolling(window=30, center=False).mean()
summary = df.describe()
```

From the commands above, three additional columns ('Daily Return', '10DMA', and '30DMA') are added to the DataFrame. The 'Daily Return' column is calculated by calling the `pct_change()` method, which calculates the percentage change between each row in the 'Adj Close' column. The '10DMA' and '30DMA' columns are calculated by creating rolling 10-day or 30-day windows on the rows in the 'Adj Close' column, and then calculating the mean on those windows.

Then the 'summary' variable is defined as 'df.describe()'. As demonstrated in the previous section, this method will display some common sample statistics whenever a user types 'summary' and hits enter in the prompt.

Using the daily return values, a user can also calculate an annualized volatility. From the command below, a standard deviation of the daily returns is first calculated and multiplied by the square root of the number of trading days in a year.

```
annual_vol = df['Daily Return'].std() * np.sqrt(df['Adj Close'].count())
```

A user can simply type in 'annual_vol' and hit enter in the prompt to query the calculated annualized volatility.

The `to_csv()` method allows a user to export the DataFrame to a csv file. Using the following sample commands, a user can export the entire DataFrame as a csv file with a file name 'NFLX_calculated.csv' or just the summary part with a file name 'NFLX_summary.csv'. The exported files will be made available in the same directory.

```
df.to_csv('NFLX_calculated.csv')
df.describe().to_csv('NFLX_summary.csv')
```

Anaconda Prompt - python

```
>>> df['Daily Return'] = df['Adj Close'].pct_change()
>>> df['10DMA'] = df['Adj Close'].rolling(window=10, center=False).mean()
>>> df['30DMA'] = df['Adj Close'].rolling(window=30, center=False).mean()
>>> summary = df.describe()
>>> summary
```

	Open	High	Low	Close	Adj Close
count	252.000000	252.000000	252.000000	252.000000	252.000000
mean	221.316428	224.245437	217.969603	221.367262	221.367262
std	59.792114	60.913089	58.302900	59.698108	59.698108
min	146.130005	147.270004	144.250000	146.169998	146.169998
25%	179.400005	181.165001	177.362499	179.219997	179.219997
50%	194.500000	196.135002	192.075005	195.000000	195.000000
75%	281.235001	286.160004	275.209992	280.477509	280.477509
max	336.299988	338.820007	331.149994	336.059998	336.059998

	Volume	Daily Return	10DMA	30DMA
count	2.520000e+02	251.000000	243.000000	223.000000
mean	8.209865e+06	0.003221	220.502037	218.735544
std	4.991033e+06	0.023119	57.915289	53.659616
min	2.160500e+06	-0.061370	150.115000	155.069000
25%	5.021075e+06	-0.008125	179.601001	176.223833
50%	6.643700e+06	0.000924	194.604001	193.716000
75%	9.669375e+06	0.014887	273.648499	271.125332
max	4.158740e+07	0.135436	328.204999	320.890333

```
>>> annual_vol = df['Daily Return'].std() * np.sqrt(df['Adj Close'].count())
>>> annual_vol
0.367007635319521
>>> df.to_csv('NFLX_calculated.csv')
>>> df.describe().to_csv('NFLX_summary.csv')
>>>
>>>
```

NFLX_calculated.csv

	A	B	C	D	E	F	G	H	I	J	K
1		Date	Open	High	Low	Close	Adj Close	Volume	Daily Retu	10DMA	30DMA
2	0	5/23/2017	157.75	158.31	156.8	157.95	157.95	3370900			
3	1	5/24/2017	158.35	158.48	157.17	157.75	157.75	2970800	-0.00127		
4	2	5/25/2017	161	164.1	160.55	163.05	163.05	8561000	0.033597		
5	3	5/26/2017	162.84	163.05	161.12	162.43	162.43	4834300	-0.0038		
6	4	5/30/2017	163.6	164.75	162.71	163.22	163.22	4828600	0.004864		
7	5	5/31/2017	163.61	164	160.74	163.07	163.07	5328900	-0.00092		
8	6	6/1/2017	163.52	163.93	161.7	162.99	162.99	3896300	-0.00049		
9	7	6/2/2017	163.42	165.36	162.8	165.18	165.18	4259100	0.013436		
10	8	6/5/2017	165.49	165.5	163.43	165.06	165.06	3875200	-0.00073		
11	9	6/6/2017	164.95	166.82	164.51	165.17	165.17	4382100	0.000666	162.587	
12	10	6/7/2017	165.6	166.4	164.41	165.61	165.61	3353100	0.002664	163.353	
13	11	6/8/2017	166.12	166.87	164.84	165.88	165.88	3695200	0.00163	164.166	
14	12	6/9/2017	166.27	166.27	154.5	158.03	158.03	10292000	-0.04732	163.664	
15	13	6/12/2017	155.3	155.53	148.31	151.44	151.44	14114500	-0.0417	162.565	
16	14	6/13/2017	154.38	155.68	150.13	152.72	152.72	8484700	0.008452	161.515	
17	15	6/14/2017	154.34	155.62	150.28	152.2	152.2	6461800	-0.0034	160.428	
18	16	6/15/2017	149.44	152.56	147.3	151.76	151.76	7319700	-0.00289	159.305	
19	17	6/16/2017	151.45	153.53	150.39	152.38	152.38	6909700	0.004085	158.025	
20	18	6/19/2017	154.29	155.58	152.41	153.4	153.4	6544300	0.006694	156.859	
21	19	6/20/2017	153.68	154.5	151.4	152.05	152.05	4878700	-0.0088	155.547	
22	20	6/21/2017	152.5	155.38	152.26	155.03	155.03	5803400	0.019599	154.489	
23	21	6/22/2017	155.13	155.2	153.7	154.89	154.89	3769200	-0.0009	153.39	
24	22	6/23/2017	155.01	158.19	153.76	158.02	158.02	6250800	0.020208	153.389	
25	23	6/26/2017	158.78	159.97	156.56	157.5	157.5	6016000	-0.00329	153.995	
26	24	6/27/2017	156.62	156.98	150.72	151.03	151.03	7424300	-0.04108	153.826	
27	25	6/28/2017	151.64	154.2	150.12	153.41	153.41	5589900	0.015758	153.947	
28	26	6/29/2017	152.82	152.82	148	150.09	150.09	7142500	-0.02164	153.78	
29	27	6/30/2017	149.76	150.71	148.42	149.41	149.41	5213300	-0.00453	153.483	
30	28	7/3/2017	149.8	150.45	145.8	146.17	146.17	3908200	-0.02169	152.76	
31	29	7/5/2017	146.58	148.26	145.58	147.61	147.61	4627800	0.009852	152.316	156.8167
32	30	7/6/2017	146.13	147.27	144.25	146.25	146.25	5486500	-0.00921	151.438	156.4267
33	31	7/7/2017	146.65	150.75	146.65	150.18	150.18	5561300	0.026872	150.967	156.1743

NFLX_summary.csv

	A	B	C	D	E	F	G	H	I	J
1		Open	High	Low	Close	Adj Close	Volume	Daily Retu	10DMA	30DMA
2	count	252	252	252	252	252	252	251	243	223
3	mean	221.3164	224.2454	217.9696	221.3673	221.3673	8209865	0.003221	220.502	218.7355
4	std	59.79211	60.91309	58.3029	59.69811	59.69811	4991033	0.023119	57.91529	53.65962
5	min	146.13	147.27	144.25	146.17	146.17	2160500	-0.06137	150.115	155.069
6	25%	179.4	181.165	177.3625	179.22	179.22	5021075	-0.00813	179.601	176.2238
7	50%	194.5	196.135	192.075	195	195	6643700	0.000924	194.604	193.716
8	75%	281.235	286.16	275.21	280.4775	280.4775	9669375	0.014887	273.6485	271.1253
9	max	336.3	338.82	331.15	336.06	336.06	41587400	0.135436	328.205	320.8903
10										

Summary

This concludes a basic introduction to the use of the Pandas package for data analysis, similar to the basic data analysis and manipulations one would perform in Microsoft Office Excel. For more information about other methods to view and manipulate data in Pandas, please refer to the [current documentation](#).