Introduction to computer programming with Python (for MGMT undergrads)

Bill Chau, CFA, FRM
Intern Analyst, Investment Research - Total Portfolio Management

CPP Investment Board ★★★★★ 5 reviews - Toronto, ON

Apply Now

Qualifications

- Pursuing an undergraduate or advanced degree in Business, Computer Science, Finance, Mathematics, Statistics, Operations Research or related field
- Students returning to school in the fall of 2019 are eligible to apply
- Strong analytical and problem solving skills
- Attention to detail, intellectual curiosity and ability to learn new concepts
- Ability to adapt to changing priorities and new challenges
- Strong interest in the financial markets, portfolio management, portfolio construction and/or asset pricing are desired; enrolment in CFA program is considered an asset
- Strong interpersonal skills with demonstrated solid verbal and written communication skills
- Demonstrated ability to collaborate and work in teams, while also being self-motivated and able to work independently
- Strong technical aptitude and experience working with databases and programming languages (e.g. Excel/ VBA, Python, SQL, large data tools such as Hadoop/Hive) is an asset
- Successful candidates must exemplify CPP Investment Board's Guiding Principles of High Performance, Integrity, and Partnership
Workshop breakdown

• Intro to Python
• Anaconda/Jupyter Notebook
• Pandas
• Matplotlib
• Quandl Financial Data API
What is Programming and Coding?

• Programming is the process of writing “programs” that a computer can execute and produce some (useful) output. Programming is a multi-step process that involves the following steps:
  • Identifying the aspects of the real-world problem that can be solved computationally
  • Identifying (the best) computational solution
  • Implementing the solution in a specific computer language
  • Testing, validating, and adjusting implemented solution.
  • While “Programming” refers to all of the above steps, “Coding” refers to step 3 only: “Implementing the solution in a specific computer language”.

• Let’s see some examples
What is Python and why should I learn it

• Python is a general purpose programming language that supports rapid development of data analytics applications. The word “Python” is used to refer to both, the programming language and the tool that executes the scripts written in Python language.

• Its main advantages are:
  • Free
  • Open-source
  • Available on all major platforms (macOS, Linux, Windows)
  • Supported by Python Software Foundation
  • Supports multiple programming paradigms
  • Has large community
  • Rich ecosystem of third-party packages

• So, why do you need Python for data analysis?
Easy to learn

"Hello, World"

- **C**
  ```c
  #include <stdio.h>
  
  int main(int argc, char ** argv)
  {
    printf("Hello, World!\n");
  }
  ```

- **Java**
  ```java
  public class Hello
  {
    public static void main(String argv[])
    {
      System.out.println("Hello, World!");
    }
  }
  ```

- **now in Python**
  ```python
  print "Hello, World!"
  ```
Reproducibility

• Reproducibility is the ability to obtain the same results using the same dataset(s) and analysis.

• Data analysis written as a Python script can be reproduced on any platform. Moreover, if you collect more or correct existing data, you can quickly and easily re-run your analysis!

• An increasing number of journals and funding agencies expect analyses to be reproducible, so knowing Python will give you an edge with these requirements.
Versatility

• Python is a versatile language that integrates with many existing applications to enable something completely amazing.

• Python can read text files, connect to databases, and many other data formats, on your computer or on the web.
Interdisciplinary and extensible

• Python provides a framework that allows anyone to combine approaches from different research (but not only) disciplines to best suit your analysis needs.
Python has a large and welcoming community

• Thousands of people use Python daily. Many of them are willing to help you through mailing lists and websites, such as Stack Overflow and Anaconda community portal.
Free and Open-Source Software (FOSS) ... and Cross-Platform

• This is important
Knowing your way around Anaconda

• **Anaconda** distribution of Python includes a lot of its popular packages, such as the IPython console, Jupyter Notebook, and Spyder IDE. Have a quick look around the Anaconda Navigator. You can launch programs from the Navigator or use the command line.

• The **Jupyter Notebook** is an open-source web application that allows you to create and share documents that allow one to easily create documents that combine code, graphs, and narrative text. **Spyder** is an Integrated Development Environment that allows one to write Python scripts and interact with the Python software from within a single interface.

• Anaconda also comes with a package manager called **conda**, which makes it easy to install and update additional packages.
Jupyter Notebook

• This is a web-based Python live-code viewer, it will allow us to experiment some basic Python syntaxes
• This is a great tool for us to practice running python codes line by line
Jupyter Notebook

• You have now created a new Python script, let’s try to use it as an ‘advanced calculator’ and run each line at a time.

```
In [ ]:
```

• Enter the following in the Input box”:
  • 2+2

• Click the ‘run’ button
Jupyter Notebook

• In the next Input box, type:

```python
In [ ]: print("Hello World")
```

• Click Run
Markdown

• Markdown is a quick way to add notes to your code
Introduction to Python built-in data types

One of the most basic things we can do in Python is assign values to variables:

```
In [ ]: text = "hamburger"
number = 52
pi_value = 3.1415
```

• Here we’ve assigned data to the variables text, number and pi_value, using the `assignment operator =`.  
• To review the value of a variable, we can type the name of the variable into the interpreter and click `run`
• Everything in Python has a type. To get the type of something, we can pass it to the built-in function `type`

```
In [ ]: type(number)
```

• The variable text is of type `str`, short for “string”. Strings hold sequences of characters, which can be letters, numbers, punctuation or more exotic forms of text (even emoji!).

• We can also see the value of something using another built-in function, `print`:

```
In [ ]: print(text)
```
Operators

• We can perform mathematical calculations in Python using the basic operators +, -, /, *, %:
  • 2 + 2
  • 6 * 7
  • 2 ** 16
  • 13 % 5
Comparison and Logic operators

• We can also use comparison and logic operators: <, >, ==, !=, <=, >= and statements of identity such as and, or, not. The data type returned by this is called a boolean.
  • 3>4
  • True and True
  • True or False
  • True and False
Sequences: Lists and Tuples

• Lists
  • **Lists** are a common data structure to hold an ordered sequence of elements. Each element can be accessed by an index. Note that Python indexes start with 0 instead of 1

```python
In [1]: fruits=['apple', 'orange', 'banana']
```

• A **for** loop can be used to access the elements in a list or other Python data structure one at a time:

```python
In [1]: for fruit in fruits:
   ...:     print(fruit)
```
• **Indentation** is very important in Python. Note that the second line in the example above is indented. In Jupyter Notebook, indentation is automatic when you press enter.

• To add elements to the end of a list, we can use the `append` method. Methods are a way to interact with an object (a list, for example). We can invoke a method using the dot . followed by the method name and a list of arguments in parentheses. Let’s look at an example using `append`:

```python
In [ ]: fruits.append("pear")
```

To find out what methods are available for an object, we can use the built-in help command:

```python
In [18]: help(fruits)
Help on list object:
class list(object)
    |     list() -> new empty list
```
Tuples

• A tuple is similar to a list in that it’s an ordered sequence of elements. However, tuples can not be changed once created (they are “immutable”). Tuples are created by placing comma-separated values inside parentheses ().

```python
In [ ]: colors=("blue", "red", "yellow")
```

• Exercise:
  • Try retrieving the value stored in position “1”
  • Try changing the value of position “2”. colors[2]=“white”
  • What error message did you get?
Dictionaries

• A **dictionary** is a container that holds pairs of objects - keys and values.

```
In [ ]: player_number={"lowry":7, "ibaka":9}
```

```
In [27]: player_number["lowry"]
```

```
Out[27]: 7
```

• Dictionaries work a lot like lists - except that you index them with **keys**. You can think about a key as a name for or a unique identifier for a set of values in the dictionary.

• Exercise: Try adding Leonard into the dictionary “player_number”
• Using for loops with dictionaries is a little more complicated:

```
In [ ]: for key, value in player_number.items():
    print(key, value)
```
Functions

• Defining a section of code as a function in Python is done using the `def` keyword. For example a function that takes two arguments and returns their sum can be defined as:

```
In [34]: def add_function(a,b):
   ...:     result = a+b
   ...:     return result

In [35]: z = add_function(2, 3)

In [36]: print(z)
```
Starting With Data

• We can automate the process of performing data manipulations in Python. It’s efficient to spend time building the code to perform these tasks because once it’s built, we can use it over and over on different datasets that use a similar format. This makes our methods easily reproducible. We can also easily share our code with colleagues and they can replicate the same analysis.
Organizing your working directory

- Using a consistent folder structure across your projects will help you keep things organized, and will also make it easy to find/file things in the future. This can be especially helpful when you have multiple projects. In general, you may wish to create separate directories for your scripts, data, and documents.

  - **data/**: Use this folder to store your raw data. For the sake of transparency and provenance, you should always keep a copy of your raw data. If you need to cleanup data, do it programmatically (i.e. with scripts) and make sure to separate cleaned up data from the raw one. For example, you can store raw data in files ./data/raw/ and clean data in ./data/clean/.

  - **documents/**: Use this folder to store outlines, drafts, and other text.

  - **scripts/**: Use this folder to store your (Python) scripts for data cleaning, analysis, and plotting that you use in this particular project.

- You may need to create additional directories depending on your project needs, but these should form the backbone of your project’s directory. For this workshop, we will need a **data/** folder to store our raw data, and we will create later a **data_output/** folder when we learn how to export data as CSV files.
Data

• For this lesson, we will be using the Global IPO data from the last 3 years.
• download the sample dataset from The BRIDGE website
• Download the file *ipo_raw.csv* and put it in your directory folder – *data/*
• We are studying the issuer, industry and listed exchanges etc.
• The dataset is stored as a .csv file: each row holds information for a single IPO
Pandas in Python

• One of the best options for working with tabular data in Python is to use the Python Data Analysis Library (a.k.a. Pandas). The Pandas library provides data structures, produces high quality plots with matplotlib and integrates nicely with other libraries that use NumPy (which is another Python library) arrays.
About Libraries

• A library in Python contains a set of tools (called functions) that perform tasks on our data. Importing a library is like getting a piece of lab equipment out of a storage locker and setting it up on the bench for use in a project. Once a library is set up, it can be used or called to perform many tasks.

• *A quick note aside that there are Python libraries like OS Library that can work with our directory structure, however, that is not our focus today.

• *Later in the workshop, we will need access to libraries like Pandas, Statsmodels, Ipython and NumPy etc.
Pandas in Python

• Python doesn’t load all of the libraries available to it by default. We have to add an `import` statement to our code in order to use library functions. To import a library, we use the syntax `import libraryName`. If we want to give the library a nickname to shorten the command, we can add `as nickNameHere`. An example of importing the pandas library using the common nickname pd is below.

```
In [ ]: import pandas as pd
```

• Each time we call a function that’s in a library, we use the syntax `LibraryName.FunctionName`. Adding the library name with a . before the function name tells Python where to find the function. In the example above, we have imported Pandas as `pd`. This means we don’t have to type out pandas each time we call a Pandas function.
Reading CSV Data Using Pandas

• We will begin by locating and reading our IPO data which are in CSV format. CSV stands for Comma-Separated Values and is a common way store formatted data. Other symbols may also be used, so you might see tab-separated, colon-separated or space separated files. It is quite easy to replace one separator with another, to match your application. The first line in the file often has headers to explain what is in each column. CSV (and other separators) make it easy to share data, and can be imported and exported from many applications, including Microsoft Excel. For more details on CSV files, see the Data Organisation in Spreadsheets lesson. We can use Pandas’ read_csv function to pull the file directly into a DataFrame.
So What’s a DataFrame?

• A DataFrame is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, factors and more) in columns. It is similar to a spreadsheet or an SQL table or the `data.frame` in R. A DataFrame always has an index (0-based). An index refers to the position of an element in the data structure.

```
In [ ]: # Note that pd.read_csv is used because we imported pandas as pd
    pd.read_csv("data/ipo_raw.csv")
```

• The above command yields the output below:
We can see that there were 5,000 rows parsed. Each row has 13 columns. The first column is the index of the DataFrame. The index is used to identify the position of the data, but it is not an actual column of the DataFrame. It looks like the read_csv function in Pandas read our file properly. However, we haven’t saved any data to memory so we can’t work with it. We need to assign the DataFrame to a variable. Remember that a variable is a name for a value, such as x, or data. We can create a new object with a variable name by assigning a value to it using =.

Let’s call the imported ipo data ipo_df:

```python
In [ ]: ipo_df = pd.read_csv("data/ipo_raw.csv")
```

Notice when you assign the imported DataFrame to a variable, Python does not produce any output on the screen. We can view the value of the ipo_df object by typing its name into the Python command prompt.
• Note: if the output is too wide to print on your narrow terminal window, you may see something slightly different as the large set of data scrolls past. You may see simply the last column of data:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>06/20/2018</td>
<td>VEA AU</td>
<td>Viva Energy Group Ltd</td>
<td>Energy</td>
<td>ASE</td>
<td>1955.930000</td>
<td>Trading</td>
<td>IPO,Primary Share Offering, Best Efforts, REG S,...</td>
</tr>
</tbody>
</table>
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ...
| 4970 | 12/18/2018 | 1686259D CN | Cuspis Capital Ltd | Financial | Venture | 0.741565 | Pending | IPO,Primary Share Offering, Best Efforts | Common | NaN | 12/18/2018 |
| 4971 | 05/10/2016 | PROVIT SS | Provide IT Sweden AB | Technology | Spotlight | 0.738380 | Trading | IPO,Primary Share Offering, Best Efforts | Common | 06/27/2016 | 05/10/2016 |

• Never fear, all the data is there, if you scroll up. Selecting just a few rows, so it is easier to fit on one window, you can see that pandas has neatly formatted the data to fit our screen:

```python
In [ ]: # The head() method displays the first several lines of a file.
ipo_df.head()
```
Data Types and Formats

• The format of individual columns and rows will impact analysis performed on a dataset read into python. For example, you can’t perform mathematical calculations on a string (text formatted data). This might seem obvious, however sometimes numeric values are read into Python as strings. In this situation, when you then try to perform calculations on the string-formatted numeric data, you get an error.

• In this lesson we will review ways to explore and better understand the structure and format of our data.
Types of Data

• How information is stored in a DataFrame or a Python object affects what we can do with it and the outputs of calculations as well. There are two main types of data that we’re explore in this lesson: numeric and text data types.
  • Numeric Data Types (int or float)
  • Text Data Type (word, sentence or a mix of word and number)
# Data type in Pandas

<table>
<thead>
<tr>
<th>Pandas Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).</td>
</tr>
<tr>
<td>int64</td>
<td>Numeric characters. 64 refers to the memory allocated to hold this character.</td>
</tr>
<tr>
<td>float64</td>
<td>Numeric characters with decimals. If a column contains numbers and NaNs (see below), pandas will default to float64, in case your missing value has a decimal.</td>
</tr>
<tr>
<td>datetime64, timedelta[ns]</td>
<td>Values meant to hold time data. Look into these for time series experiments.</td>
</tr>
</tbody>
</table>
Exploring the IPO Data

• Again, we can use the type function to see what kind of thing ipo_df is:

```python
In [14]: type(ipo_df)
```

```python
Out[14]: pandas.core.frame.DataFrame
```

• As expected, it’s a DataFrame (or, to use the full name that Python uses to refer to it internally, a pandas.core.frame.DataFrame).

• What kind of things does `ipo_df` contain? DataFrames have an attribute called `dtypes` that answers this:

```python
In []: ipo_df.dtypes
```
Useful Ways to View DataFrame objects in Python

- There are many ways to summarize and access the data stored in DataFrames, using attributes and methods provided by the DataFrame object.

- To access an attribute, use the DataFrame object name followed by the attribute name `df_object.attribute`. Using the DataFrame `ipo_df` and attribute `columns`, an index of all the column names in the DataFrame can be accessed with `ipo_df.columns`.

- Methods are called in a similar fashion using the syntax `df_object.method()`. As an example, `ipo_df.head()` gets the first few rows in the DataFrame `ipo_df` using the `head()` method. With a method, we can supply extra information in the parents to control behavior.
Exercise

• Using our DataFrame `ipo_df`, try out the attributes & methods below to see what they return.
  • `ipo_df.columns`
  • `ipo_df.shape` Take note of the output of shape - what format does it return the shape of the DataFrame in?
  • `ipo_df.head()` Also, what does `ipo_df.head(15)` do?
  • `ipo_df.tail()`
We’ve read our data into Python. Next, let’s perform some quick summary statistics to learn more about the data that we’re working with. We might want to know how many IPOs were listed in each exchange, or how much of money were raised in each industry. We can perform summary stats quickly using groups. But first we need to figure out what we want to group by.

Let’s begin by exploring our data:

In [ ]: # Look at the column names
ipo_df.columns
Let’s get a list of all the exchanges. The `pd.unique` function tells us all of the unique values in the "Primary Exchange 'column."

```python
In [25]: pd.unique(ipo_df['Primary Exchange'])
```

```
Out[25]: array(['Tokyo', 'Hong Kong', nan, 'Xetra', 'Shanghai', 'New York',
'BrsaItaliana', 'NASDAQ GS', 'Korea SE', 'SIX Swiss Ex', 'London',
'Vienna', 'ASE', 'Natl India', 'Singapore', 'B3 Day',
'London Intl', 'Mexico SE', 'Ho Chi Minh', 'Bangkok', 'DFM',
'Toronto', 'EN Paris', 'Johannesburg', 'EN Amsterdam', 'Warsaw',
'Helsinki', 'B3 Soma', 'OTC US', 'BYMA Floor', 'Shenzhen',
'Soc.Bol SIBE', 'Frankfurt', 'Oslo', 'KOSDAQ', 'Abu Dhabi',
'Bursa Malays', 'Prague SE', 'Stockholm', 'Ljubljana', 'Qatar',
'Dublin', 'Istanbul', 'EN Brussels', 'Copenhagen', 'NASDAQ CM',
'Luxembourg', 'Philippines', 'Sant. Comerc', 'Norway OTC',
'MICEX Main', 'Indonesia', 'FN Stockholm', 'Reykjavik',
'NASDAQ GM', 'BSE India', 'Accra', 'Hanoi UPCoM', 'Saudi Arabia',
'Kuwait', 'Dar E Slm SE', 'Bucharest', 'NYSEAmerican', 'Tel Aviv',
'Casablanca', 'SEM', 'Tallinn', 'EGX', 'NZX', 'OTC BB',
'Montevideo', 'Nasdaq Dubai', 'Taipei', 'BRVM Region',
'Canadian Sec', 'Budapest', 'Agts Neo Lit', 'Muscat', 'Taiwan',
'Lima', 'Dhaka', 'Bulgaria SE', 'Venture', 'Uganda',
'KazkhstnStEx', 'Maldives S E', 'Pakistan', 'BIVA', 'FN Finland',
'Bahrain Brse', 'Gaborone', 'Belgrade SE', 'Kingston', 'Windhoek',
], dtype='|S9')
```
Exercise

• Create a list of unique exchanges ("Primary Exchange") found in the IPO data. Call it `exchange_names`. How many unique exchanges are there in the data?

• How many unique IPOs are in the data?

• What is the difference between `len(exchange_names)` and `ipo_df["Primary Exchange"].nunique()`?
Groups in Pandas

• We often want to calculate summary statistics grouped by subsets or attributes within fields of our data. For example, we might want to calculate the average money raised of all IPOs per exchange.

• We can calculate basic statistics for all records in a single column using the syntax below:

  ```python
  In [36]: ipo_df["Offer Size (M)"].describe()
  Out[36]:
  count      5000.000000
  mean      122.843068
  std       465.433029
  min       0.658562
  25%       6.318233
  50%      21.655050
  75%      99.379625
  max     21140.100000
  Name: Offer Size (M), dtype: float64
  ```
• We can also extract one specific metric if we wish:

```python
In [28]: ipo_df['Offer Size (M)'].min()
ipo_df['Offer Size (M)'].max()
ipo_df['Offer Size (M)'].mean()
ipo_df['Offer Size (M)'].std()
ipo_df['Offer Size (M)'].count()
```

• But if we want to summarize by one or more variables, for example ‘Primary Exchange’, we can use **Pandas’ .groupby method**. Once we’ve created a groupby DataFrame, we can quickly calculate summary statistics by a group of our choice.

```python
In [ ]: # Group data by Exchange
grouped_data = ipo_df.groupby("Primary Exchange")
```

• The **pandas function describe** will return descriptive stats including: mean, median, max, min, std and count for a particular column in the data. Pandas’ describe function will only return summary values for columns containing numeric data.

```python
In [ ]: # Summary statistics for all numeric columns by exchange
grouped_data.describe()
# Provide the mean for each numeric column by exchange
grouped_data.mean()
```
• The **groupby** command is powerful in that it allows us to quickly generate summary stats.

• Exercise:
  • How many recorded IPOs are in the Financial sector
  • What happens when you group by two columns using the following syntax and then grab mean values:
    • grouped_data2 = ipo_df.groupby([‘Industry Sector’,’Primary Exchange’])
    • grouped_data2.mean()
  • Summarize Offer Size(M) values for each Exchange in your data.
    *HINT: you can use the following syntax to only create summary statistics for one column in your data by_exchange[Offer Size(M)’].describe()
Quickly Creating Summary Counts in Pandas

• Let’s next count the number of IPOs for each Industry Sector. We can do this in a few ways, but we’ll use `groupby` combined with a `count()` method.

```python
In [54]:
# Count the number of IPO by Industry Sector
ipo_counts = ipo_df.groupby('Industry Sector')['Issuer Ticker'].count()
print(ipo_counts)
```

• Or, we can also count just the rows that are in a specific Industry Sector:

```python
In [56]:
ipo_counts = ipo_df.groupby('Industry Sector')['Issuer Ticker'].count()['Energy']
print(ipo_counts)
```

195
Indexing, Slicing and Subsetting DataFrames in Python

• In the first episode of this lesson, we read a CSV file into a pandas’ DataFrame. We learned how to:
  • save a DataFrame to a named object
  • perform basic math on data
  • calculate summary statistics
  • create plots based on the data we loaded into pandas

• In this lesson, we will explore ways to access different parts of the data using:
  • indexing
  • slicing
  • subsetting
Loading the data

• We will continue to use the IPO dataset that we worked with in the last episode. Let’s reopen and read in the data again:

```
In [ ]:
# Make sure pandas is loaded
import pandas as pd

# Read in the survey CSV
ipo_df = pd.read_csv("data/ipo_raw.csv")
```

*download the *ipo_raw.csv* file from Libguide and place it in the */data* folder*
Indexing and Slicing in Python

• We often want to work with subsets of a DataFrame object. There are different ways to accomplish this including: using labels (column headings), numeric ranges, or specific x,y index locations.
Selecting data using Labels (Column Headings)

• We use square brackets [ ] to select a subset of a Python object. For example, we can select all data from a column named “Issuer Name” from the ipo_df DataFrame by name.

```
In [ ]: # TIP: use the .head() method we saw earlier to make output shorter
       # Select a ‘subset’ of the data using the column name
       ipo_df['Issuer Name']
```

• We can also create a new object that contains only the data within the “Issuer Name” column as follows:

```
In [ ]: ipoIssuer = ipo_df['Issuer Name']
```
Exercise:

• We can pass a list of column names too, as an index to select columns in that order. This is useful when we need to reorganize our data.

• **NOTE:** If a column name is not contained in the DataFrame, an exception (error) will be raised.

```python
In [ ]: # Select the issuer and ticker columns from the DataFrame
ipo_df[['Issuer Name', 'Issuer Ticker']]

# What happens when you flip the order?
ipo_df[['Issuer Ticker', 'Issuer Name']]

# What happens if you ask for a column that doesn't exist?
ipo_df['Price']
```

• Python tells us what type of error it is in the traceback, at the bottom it says KeyError: 'Price' which means that Price is not a valid column name (nor a valid key in the related Python data type dictionary).
Extracting Range based Subsets: Slicing

• Reminder: Python uses 0-based indexing. This means that the first element in an object is located at position 0

• Let’s create an list of numbers

```python
In [ ]: # Create a list of numbers:
a = [1, 2, 3, 4, 5]
```

• What value does the code below return?
  • a[0]

• How about this:
  • a[5]

• In the example above, calling a[5] returns an error. Why is that?

• What about?
  • a[len(a)]
Slicing Subsets of Rows in Python

• Slicing using the [] operator selects a set of rows and/or columns from a DataFrame. To slice out a set of rows, you use the following syntax: `data[start:stop]`. When slicing in pandas the start bound is included in the output. The stop bound is one step BEYOND the row you want to select. So if you want to select rows 0, 1 and 2 your code would look like this:

```python
In [ ]:  # Select rows 0, 1, 2 (row 3 is not selected)
    ipo_df[0:3]

In [ ]:  # Select the first 5 rows (rows 0, 1, 2, 3, 4)
    ipo_df[:5]

    # Select the last element in the list
    # (the slice starts at the last element, and ends at the end of the list)
    ipo_df[-1:]
```
Subsetting Data using Criteria

• We can also select a subset of our data using criteria. For example, we can select all rows that have a Industry Sector of Energy:

```python
In [ ]: ipo_df[ipo_df['Industry Sector'] == "Energy"]
```

• Or we can select all rows that are not Energy:

```python
In [ ]: ipo_df[ipo_df['Industry Sector'] != "Energy"]
```

• We can define sets of criteria too:

```python
In [ ]: ipo_df[(ipo_df['Offer Size (M)']>=100) & (ipo_df['Offer Size (M)']<=500)]
```
Exercise:

• Select a subset of rows in the ipo_df DataFrame that has Offer size (M) over 1,000M and that is traded in London. How many rows did you end up with? What did your neighbor get?

• Select a subset of rows in the ipo_df DataFrame that is traded in London within the Energy Sector. How many rows did you end up with? What did your neighbor get?

*Alternatively, you may use the function `.isin`
Datetime function

• Let’s take a look at the ipo_df data type again. Column “Announced Date” is stored as object (i.e., string).
• It is much easier to sort and manipulate if we can convert it into datetime format.

```
In [28]: ipo_df['Announced Date'] = pd.to_datetime(ipo_df['Announced Date'])
```

• Now you can group data by days, months or year of your choice!!
Quick & Easy Plotting Data Using Pandas

```
In [66]: ipo_counts = ipo_df.groupby('Industry Sector')['Issuer Ticker'].count()
   print(ipo_counts)

Industry Sector          count
Basic Materials           293
Communications           333
Consumer, Cyclical       604
Consumer, Non-cyclical   1110
Diversified              175
Energy                   195
```

```
In [19]: %matplotlib inline
```

```
In [68]: ipo_counts.plot(kind='bar');
```
Exercise

• Create a plot of total offer size across all industry sectors.
• Create a plot of mean offer size across all exchanges.
What is API?

• Application Programming Interface
• Basically an interface that allows 2 independent software to integrate
• Commonly used in databases, or some functional programming
Quandl API

• Install Quandl Package (use Anaconda GUI)

• You may refer to the document here:  
  https://github.com/quandl/quandl-python

• You will also need to know what you can get out of your API


• Test drive:

In [28]:  import quandl  
quandl.ApiConfig.api_key = 'ZKCfbtZ6kDPxiPBzIbg'  
data = quandl.get('EOD/TSLA')
Quandl API

• Let’s take a closer look at the dataframe
  • Remember these commands?
    • Data.head()/data.tail()
    • Data.shape
    • Data.columns
    • Data.dtypes
    • Data.describe

• Notice that Date is not a column?
  • Date is an index!
Slicing Subsets of Rows in Python

• Slicing using the [] operator selects a set of rows and/or columns from a DataFrame. To slice out a set of rows, you use the following syntax: `data[start:stop]`. When slicing in pandas the start bound is included in the output. The stop bound is one step BEYOND the row you want to select.

```
In [ ]:
# Select the first 5 rows (rows 0, 1, 2, 3, 4)
msft_df[:5]

# Select the last element in the list
# (the slice starts at the last element, and ends at the end of the list)
msft_df[-1:]
```
Joining DataFrames

• Now let’s pull EOD price data from 2 different stock

```python
In [57]:
gold = quandl.get('EOD/GLD')
sp500 = quandl.get('EOD/SPY')
```

• They now belong to 2 data frame and if we want to do some statistical analysis on them, we will have to combine them into one dataframe
Joining Two DataFrames

• One way to combine DataFrames is to use columns in each dataset that contain common values (a common unique id). Combining DataFrames using a common field is called “joining”. The columns containing the common values are called “join key(s)”.

• In our stock example, date will be the common value
Inner joins

- The most common type of join is called an *inner join*. An inner join combines two DataFrames based on a join key and returns a new DataFrame that contains only those rows that have matching values in both of the original DataFrames.

- Inner joins yield a DataFrame that contains only rows where the value being joins exists in BOTH tables. An example of an inner join, adapted from this page is below:
Inner join

• The pandas function for performing joins is called `merge` and an Inner join is the default option:

```python
import pandas as pd
merged_inner = pd.merge(left=gold, right=sp500, left_on='Date', right_on='Date')
# What's the size of the output data?
merged_inner.shape
```
Inner joins: Caution!

- The result of an inner join of **gold** and **sp500** is a new DataFrame that contains the combined set of columns from **gold** and **sp500**. It *only* contains rows that have **Dates** that are the same in both the gold and sp500 DataFrames. In other words, if a row in gold has a value of Date that does *not* appear in the Date column of sp500, it will not be included in the DataFrame returned by an inner join. Similarly, if a row in sp500 has a value of Date that does *not* appear in the Date column of gold, that row will not be included in the DataFrame returned by an inner join.
Inner joins: Caution!

- The two DataFrames that we want to join are passed to the merge function using the left and right argument. The `left_on='species'` argument tells merge to use the `species_id` column as the join key from `survey_sub` (the left DataFrame). Similarly, the `right_on='species_id'` argument tells merge to use the `species_id` column as the join key from `species_sub` (the right DataFrame). For inner joins, the order of the left and right arguments does not matter.
Inner joins: Caution!

• The result merged_inner DataFrame contains all of the columns from gold (Open, High, Low, Close, etc.) as well as all the columns from sp500 (Open, High, Low, Close, etc).

• Notice that merged_inner has fewer rows than sp500. This is an indication that there were rows in good with value(s) for Dates that do not exist as value(s) for Dates in gold.
Matplotlib package

• **Matplotlib** is a Python package that is widely used throughout the scientific Python community to create high-quality and publication-ready graphics. It supports a wide range of raster and vector graphics formats including PNG, PostScript, EPS, PDF and SVG.

• Check out all the examples here: [https://matplotlib.org/gallery/index.html](https://matplotlib.org/gallery/index.html)
Exercise

• Let’s plot the timeseries with gold x sp500

```python
In [83]: import matplotlib
import matplotlib.pyplot as plt

In [95]: plt.plot(merged_inner.index, merged_inner["Close_x"], label="GOLD")
plt.plot(merged_inner.index, merged_inner["Close_y"], label="SP500")
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price(USD)')
plt.title("Gold and SP500 price chart")
plt.show()
```
Saving matplotlib figures

• Once satisfied with the resulting plot, you can save the plot with the `.savefig(*args)` method from matplotlib:

```python
In [ ]:
plt.plot(merged_inner.index, merged_inner["Close_x"], label="GOLD")
plt.plot(merged_inner.index, merged_inner["Close_y"], label="SP500")
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price(USD)')
plt.title("Gold and SP500 price chart")
plt.savefig("my_plot_name.png")
plt.show()
```

• Tips: Make sure you have `plt.savefig()` before `plt.show()`. Because `plt.show()` will reset your plot after it is run