# Application Programming Interface

Estimated time needed: **15** minutes

## Objectives

After completing this lab you will be able to:

- Create and Use APIs in Python

### Introduction

An API lets two pieces of software talk to each other. Just like a function, you don't have to know how the API works only its inputs and outputs. An essential type of API is a REST API that allows you to access resources via the internet. In this lab, we will review the Pandas Library in the context of an API, we will also review a basic REST API

## Table of Contents



In [1]:
```
!pip install pycoingecko
!pip install plotly
!pip install mplfinance
```

```
Collecting pycoingecko
  Downloading pycoingecko-2.2.0-py3-none-any.whl (8.3 kB)
Requirement already satisfied: requests in c:\python\lib\site-packages (from pycoingecko) (2.25.1)
Requirement already satisfied: idna<3,>=2.5 in c:\python\lib\site-packages (from requests->pycoingecko) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\python\lib\site-packages (from requests->pycoingecko) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\python\lib\site-packages (from requests->pycoingecko) (2020.12.5)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\python\lib\site-packages (from requests->pycoingecko) (1.26.4)
Installing collected packages: pycoingecko
Successfully installed pycoingecko-2.2.0
Collecting plotly
  Downloading plotly-5.6.0-py2.py3-none-any.whl (27.7 MB)
Collecting tenacity>=6.2.0
  Downloading tenacity-8.0.1-py3-none-any.whl (24 kB)
Requirement already satisfied: six in c:\python\lib\site-packages (from plotly) (1.15.0)
Installing collected packages: tenacity, plotly
Successfully installed plotly-5.6.0 tenacity-8.0.1
Collecting mplfinance
  Downloading mplfinance-0.12.8b9-py3-none-any.whl (70 kB)
Requirement already satisfied: matplotlib in c:\python\lib\site-packages (from mplfinance) (3.3.4)
Requirement already satisfied: pandas in c:\python\lib\site-packages (from mplfinance) (1.2.4)
Requirement already satisfied: python-dateutil>=2.1 in c:\python\lib\site-packages (from matplotlib->mplfinance) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\python\lib\site-packages (from matplotlib->mplfinance) (2.
4.7)
Requirement already satisfied: numpy>=1.15 in c:\python\lib\site-packages (from matplotlib->mplfinance) (1.20.1)
Requirement already satisfied: pillow>=6.2.0 in c:\python\lib\site-packages (from matplotlib->mplfinance) (8.2.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\python\lib\site-packages (from matplotlib->mplfinance) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\python\lib\site-packages (from matplotlib->mplfinance) (0.10.0)
Requirement already satisfied: six in c:\python\lib\site-packages (from cycler>=0.10->matplotlib->mplfinance) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in c:\python\lib\site-packages (from pandas->mplfinance) (2021.1)
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.8b9
```

## Pandas is an API

Pandas is actually set of software components , much of which is not even written in Python.

In [2]:
```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.offline import plot
import matplotlib.pyplot as plt
import datetime
from pycoingecko import CoinGeckoAPI
from mplfinance.original_flavor import candlestick2_ohlc
```

You create a dictionary, this is just data.

In [3]:
```
dict_={'a':[11,21,31],'b':[12,22,32]}
```

When you create a Pandas object with the Dataframe constructor in API lingo, this is an "instance". The data in the dictionary is passed along to the pandas API. You then use the dataframe to communicate with the API.

In [4]:
```
df=pd.DataFrame(dict_)
type(df)
```

Out[4]:
```
pandas.core.frame.DataFrame
```


When you call the method head the dataframe communicates with the API displaying the first few rows of the dataframe.

In [5]:
```
df.head()
```

Out[5]:

| | a | b |
|---|---|---|
| 0 | 11 | 12 |
| 1 | 21 | 22 |
| 2 | 31 | 32 |

When you call the method mean,the API will calculate the mean and return the value.

In [6]:
```
df.mean()
```

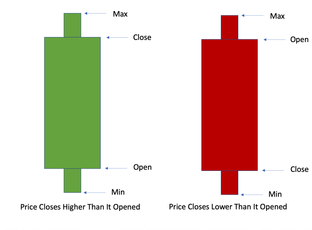
Out[6]:
```
a    21.0
b    22.0
dtype: float64
```

## REST APIs

Rest API's function by sending a request, the request is communicated via HTTP message. The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service or resource to perform. In a similar manner, API returns a response, via an HTTP message, this response is usually contained within a JSON.

In cryptocurrency a popular method to display the movements of the price of a currency.


Here is a description of the candle sticks.




In this lab, we will be using the CoinGecko API to create one of these candlestick graphs for Bitcoin. We will use the API to get the price data for 30 days with 24 observation per day, 1 per hour. We will find the max, min, open, and close price per day meaning we will have 30 candlesticks and use that to generate the candlestick graph. Although we are using the CoinGecko API we will use a Python client/wrapper for the API called PyCoinGecko. PyCoinGecko will make performing the requests easy and it will deal with the enpoint targeting.

Lets start off by getting the data we need. Using the `get_coin_market_chart_by_id(id, vs_currency, days)`. `id` is the name of the coin you want, `vs_currency` is the currency you want the price in, and `days` is how many days back from today you want.

In [7]:
```
cg = CoinGeckoAPI()
bitcoin_data = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currency='usd', days=30)
```

In [8]:
```
type(bitcoin_data )
```

Out[8]:
```
dict
```

The response we get is in the form of a JSON which includes the price, market caps, and total volumes along with timestamps for each observation. We are focused on the prices so we will select that data.

In [9]:
```
bitcoin_price_data = bitcoin_data['prices']

bitcoin_price_data[0:5]
```

Out[9]:
```
[[1645203666189, 40237.83337042184],
 [1645207240251, 40104.80427956735],
 [1645210855149, 40205.12145578287],
 [1645214529279, 40258.11567800297],
 [1645218117411, 40054.62360244416]]
```

Finally lets turn this data into a Pandas DataFrame.

In [10]:
```
data = pd.DataFrame(bitcoin_price_data, columns=['TimeStamp', 'Price'])
```

Now that we have the DataFrame we will convert the timestamp to datetime and save it as a column called `Date`. We will map our `unix_to_datetime` to each timestamp and convert it to a readable datetime.

In [11]:
```
data['date'] = data['TimeStamp'].apply(lambda d: datetime.date.fromtimestamp(d/1000.0))
```

Using this modified dataset we can now group by the `Date` and find the min, max, open, and close for the candlesticks.

In [12]:
```
candlestick_data = data.groupby(data.date, as_index=False).agg({"Price": ['min', 'max', 'first', 'last']})
```

Finally we are now ready to use plotly to create our Candlestick Chart.

In [13]:
```
fig = go.Figure(data=[go.Candlestick(x=candlestick_data['date'],
                open=candlestick_data['Price']['first'],
                high=candlestick_data['Price']['max'],
                low=candlestick_data['Price']['min'],
                close=candlestick_data['Price']['last'])
                ])

fig.update_layout(xaxis_rangeslider_visible=False)

fig.show()
```


## Authors:

## Change Log

| Date (YYYY-MM-DD) | Version | Changed By | Change Description |
|---|---|---|---|
| 2020-11-23 | 3.0 | Azim Hirjani | New API |
| 2020-09-09 | 2.1 | Malika Singla | Spell Check |
| 2020-08-26 | 2.0 | Lavanya | Moved lab to course repo in GitLab |



In [ ]:
```
```