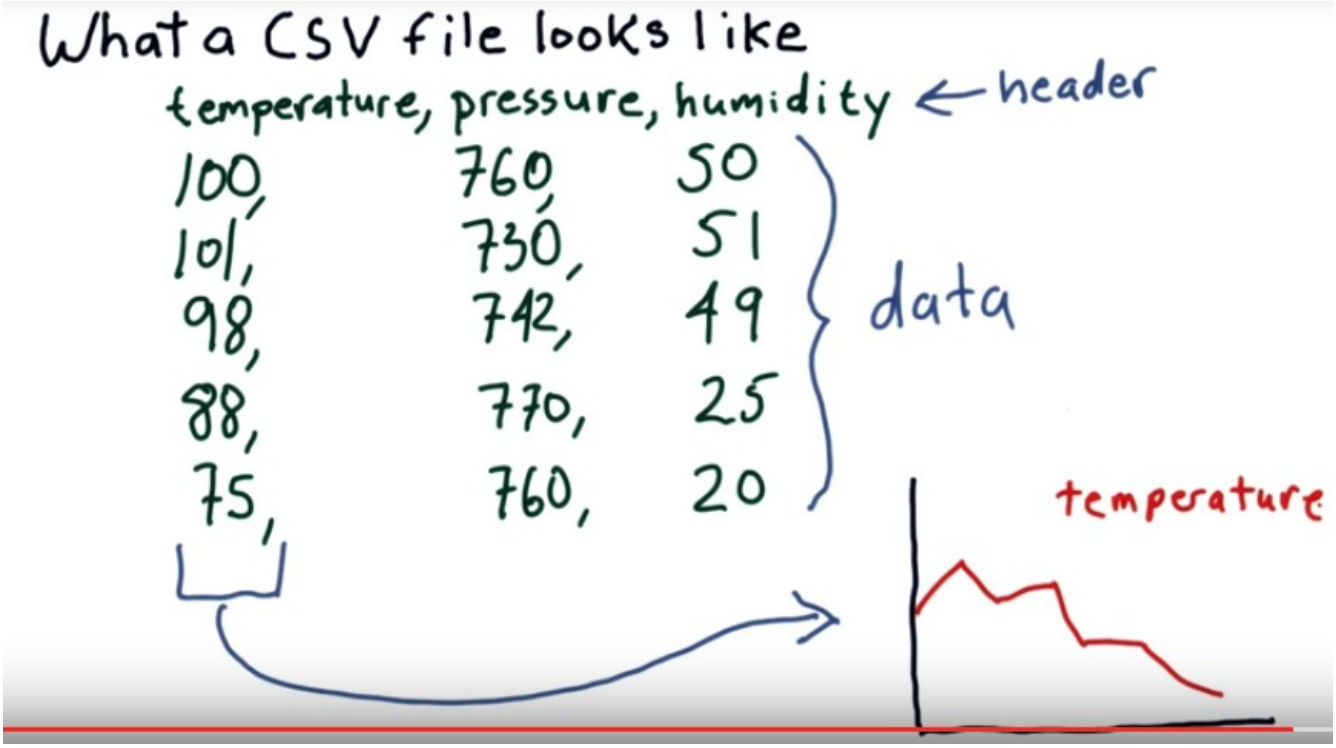
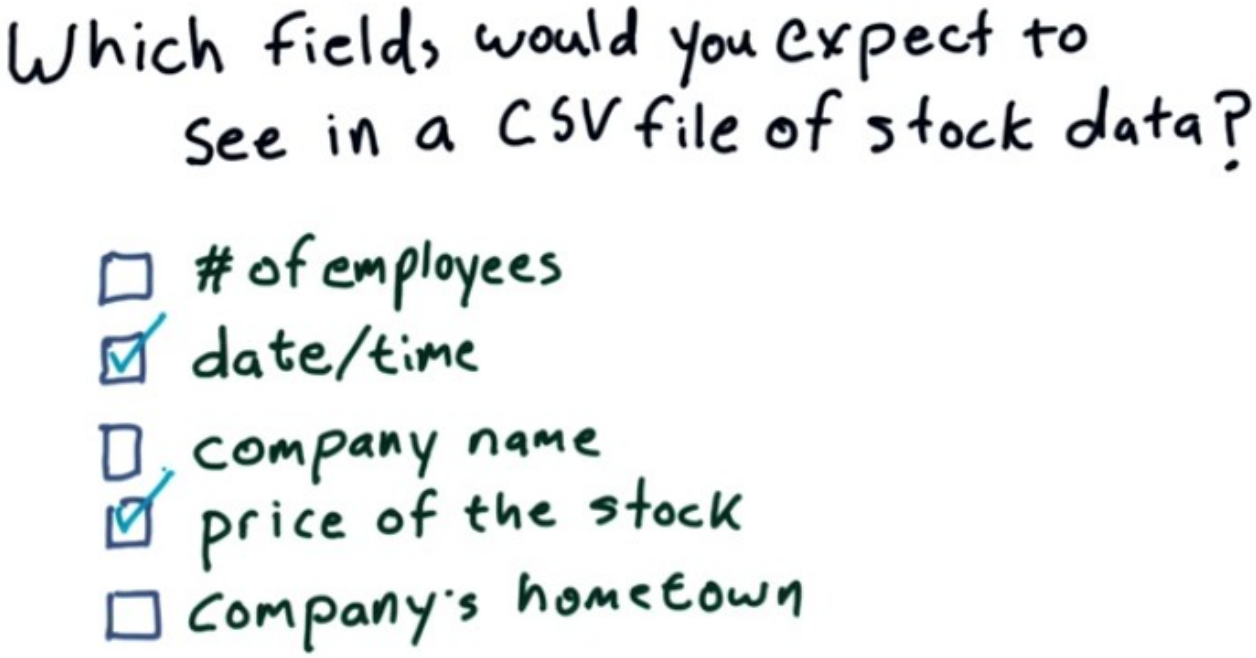
1. Hi I'm Tucker. >> And I'm Dev. >> Welcome to this mini course, Manipulating Financial Data in Python. Our goal is to give you a quick introduction to the skills you'll need to work with financial data in Python. Now, some people complain about our choice of Python for financial applications. >> I don't agree with those people. Python allows you to quickly prototype algorithms while also providing computational speed. It has a number of features. Firstly, it has strong scientific libraries. Second, it is strongly maintained. It is also fast if you can stick to metrics notation because lower levels are returned in C. >> Some other potential languages that might have made sense for this course include R and MATLAB. Which are themselves also great languages for financial data. But we've chosen Python and we'll be using this book Python for Finance in the course. Look for readings assigned in the course outline.



2. Our objective in this class is to get you up and running quickly, to show you some real data, and some code you can use to view it and manipulate it. And just give you a feeling that you know what's going on. So we're going to take you from examples of raw data all the way to visualization. Now let's get started with the data. In this class we're going to work almost entirely with data that comes from CSV files. CSV files are plain text. The C stands for comma, S stands for separated, V is values. So comma-separated values. Let me show you an example of what might be in a CSV file. Most CVS files start with a header line. In this example, this is a CSV file that's telling us about weather conditions. So we've got temperature, pressure and humidity and that tells us what information is in the columns that are to follow. Following our header line we have lines or rows of data. So these numbers here make up our data. Again, header row and rows of data, where each data element is separated by a comma. Now the files that we're going to be working with, the stock data files, have thousands of lines and many more columns as well. Our objective for this lesson is to show you how to read in data like this, focus, say, on one column or another, and create a plot from that data. Okay, so now you've seen an example with weather. Let's start thinking about stocks.

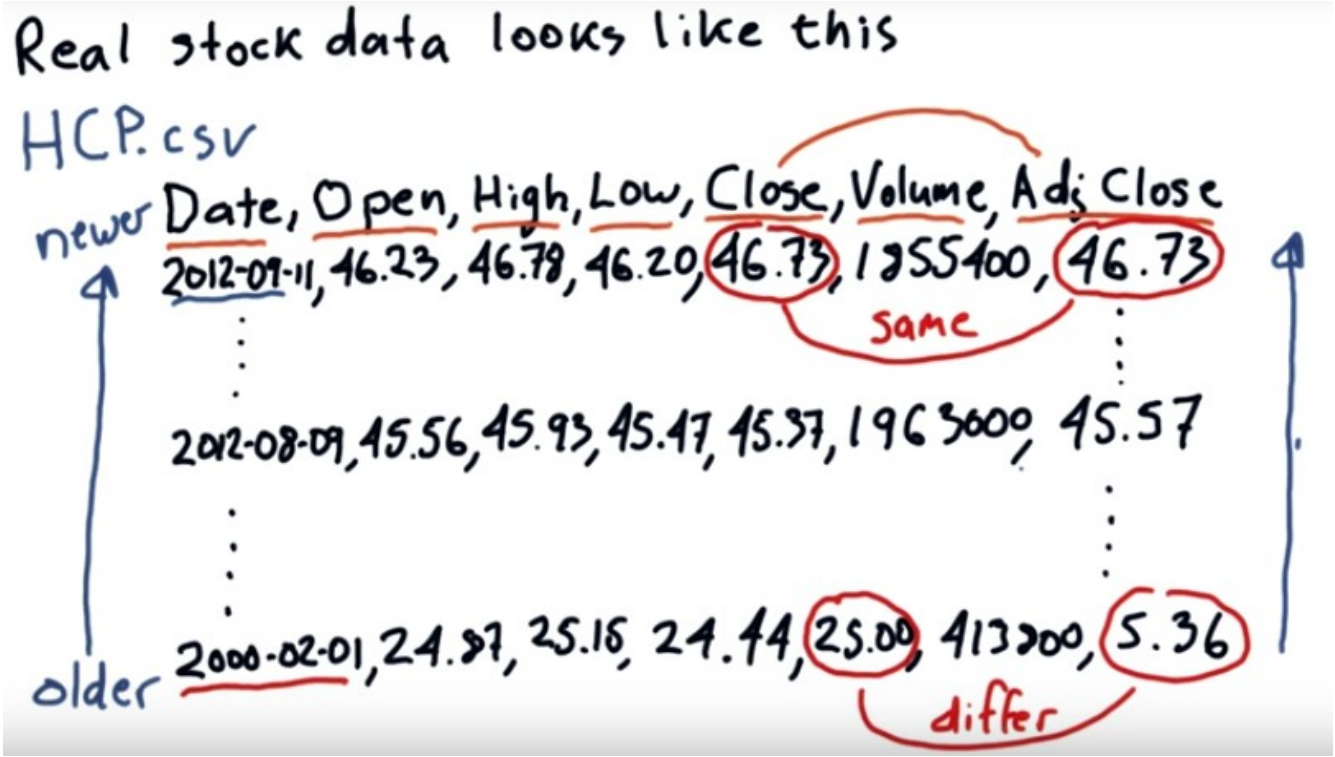
3. Which fields or items in a header row, would you expect to see in a comma separated value file of

stock data? So here are the options. Number of employees for the company, date/time, company name, price of the stock, company's hometown. Good luck.



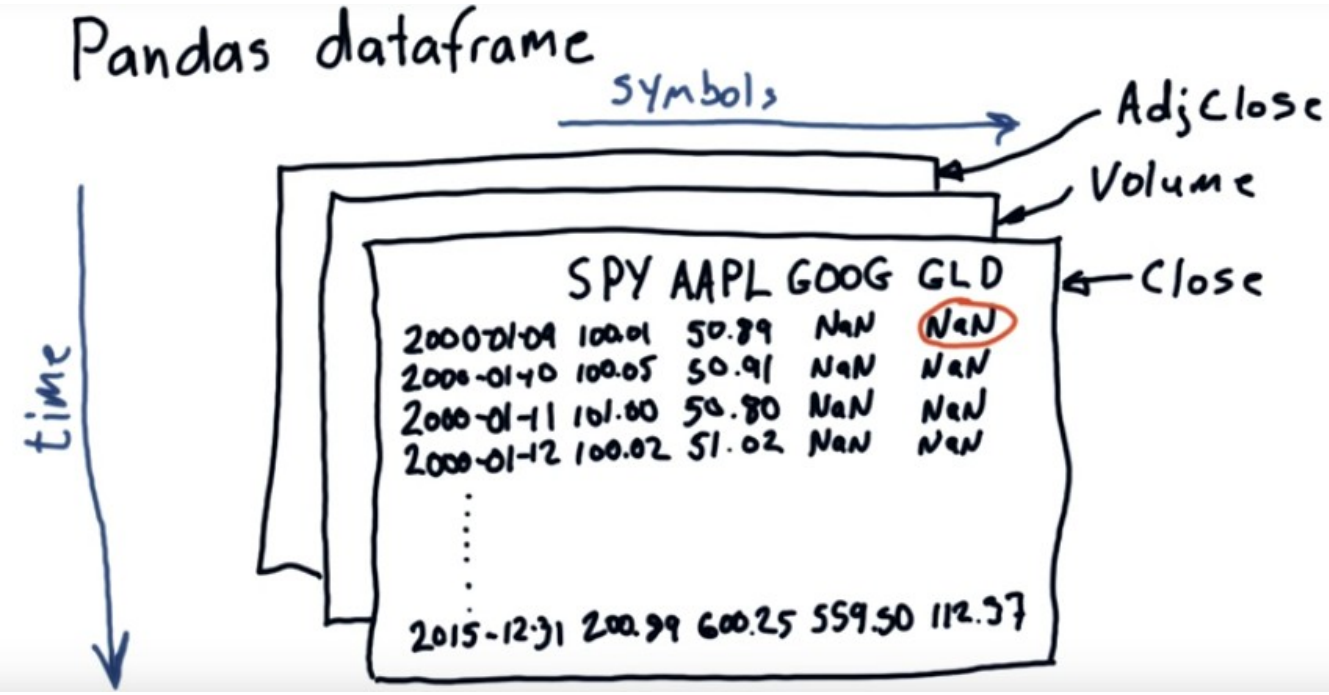
stock: 股票

4. The correct answers are date/time and price of the stock. Let me mention a couple of reasons why some of these others aren't correct answers. Sure, the company name is important to know, but it doesn't change over time, so there's no need to allocate space for this information every day over time. Company's hometown, same thing. Number of employees can be important data and actually it's Information that is sometimes provided via proprietary data feeds, but it's not something you typically find in a historical data file of stock prices.



5. Okay. So, let's take a look at some real stock data. We provide for you in this class hundreds of CSV files that represent the prices of stocks over time. Here's an example from one of those files that you provided. This is data from the file HCP.csv. So, here is our header row and here is the information that you'll find in one of these files. So, Date, which date is the information for? Open, this is the price that the stock opened at. In other words, in the morning, when trading on the exchange began, that was the first price of the day for that stock. High, throughout the day, what was the very high price, what was the very low price, and at which price did the stock close? So when we reached 4 o'clock, what was the final price? Volume, that's how many shares of the stock traded altogether on that day. And finally, this value, adjusted close, which is a little bit different from close. And this is something we cover in the next course where we talk about finance. I'll talk about it a little bit here, as well. But let me delay talking about it for a moment. Okay. I fleshed out this data a little bit. First thing I want you to notice is that the dates start with most recent dates, and then as you go forward into the file, you find older dates. So, what that means, more or less, is that we sort of go backwards through time in these files. Now, this is a feature, if you will, of data from Yahoo, and that's where we got our data for this class. Thanks very much, Yahoo. And this is just what a real one of those files look like. Now later when we read the data in, we managed to get it in the right order, and Dave will tell you a little bit more about that. Now, I had talked a little bit earlier about Adjusted Close and Close. Let me tell you a bit more about what that means. Now Close in this data is the actual price that was reported at the exchange when the stock closed for that day. Adjusted Close is a number that the data provider generates for us. And it's adjusted, as the name implies, for certain things like stocks, splits, and dividend payments. Now, on the current day, let's pretend for the moment that we're in 2012, adjusted close and close are always the same. However, as we go back in time, we eventually see that adjusted close and close differ. So if we go all the way back to the year 2000, we'll note that the actual price the stock closed at was $25, but this adjusted price was only $5.36. Now, what you can observe from that is as we go forward in time, if we had purchased this stock back in 2000 and held it to 2012, what are we looking at there? About eight or nine time return over those 12 years, so 800 to

900% return. If you looked only at just the actual price on the market, it's only a factor of about two, but this adjusted close reflects things like I said, like dividend payments, and splits, and so on. So that's what is in an actual stock CSV file. And this is the data that we're going to be working with throughout this course and the next two parts of the course.



6. We're going to make heavy use of a library called Pandas. This library was created by Wes McKinney at a hedge fund call AQR. It's used at many hedge funds and by many people in the finance industry. One of the key components of Pandas is something called the dataframe. And I'm going to show you a little bit about what that looks like. So this is the basic layout of a dataframe. We have our symbols along the top, so our columns represent symbols in the stock market. Like, SPY which is an ETF representing the S&P 500 (SPY 即 500 強 , 可以代表大多數公司 , 故可以用它來做 reference), AAPL the symbol for Apple, GOOG for Google, GLD for Gold. And the rows are the dates over time, so we go back as far as 2000, then come all the way up to 2015. So again, symbols from left to right, one column for each symbol, and time coming down like this. So here's our dataframe fleshed out with a little bit of data. I made up some of these numbers, so it's not intended to be gospel truth, but notice how we have, let's say this is closing prices. So we see these numbers for SPY, Apple, Google, and GLD. Now, there are some special or unusual values here. NaN, that stands for not a number, and that's Python's way of saying hey, I don't know, I don't have information for this. The reason you see those values here is, back in 2000, Google did not exist as a publicly traded company, and neither did the ETF GLD. Now these NaN values can cause problems, and we'll be talking about those in a later session. Now as I said, this might represent closing prices, but Pandas can also handle additional data in a sort of three dimensional sense. So you can have a dataframe that represents, again in columns, our particular symbols, and in rows, dates. This one can be close, we can have another one that has, for the same stocks and the same dates, volume on those dates, and adjusted close, and so on. So Pandas is a very flexible way to read in, manipulate, and plot data. Now I've shown you, kind of at a high level, what this data looks like and what Pandas looks like. I'm going to hand it over to Dave now, and she's going to show you some real live examples with Pandas. She's going to show you how to read this data in, and plot it and so on. So here's over to you, Dave.

```
Date,Open,High,Low,Close,Volume,Adj Close
2012-09-12,666.85,669.90,656.00,669.79,25410600,669.79
2012-09-11,665.11,670.10,656.50,660.59,17987400,660.59
2012-09-10,680.45,683.29,662.10,662.74,17428500,662.74
2012-09-07,678.05,682.48,675.77,680.44,11773800,680.44
2012-09-06,673.17,678.29,670.80,676.27,13971300,676.27
2012-09-05,675.57,676.35,669.60,670.23,12013400,670.23
2012-09-04,665.76,675.14,664.50,674.97,13139000,674.97
2012-08-31,667.25,668.60,657.25,665.24,12082900,665.24
2012-08-30,670.64,671.55,662.85,663.87,10810700,663.87
2012-08-29,675.25,677.67,672.60,673.47,7243100,673.47
2012-08-28,674.98,676.10,670.67,674.80,9550600,674.80
2012-08-27,679.99,680.87,673.54,675.68,15250300,675.68
2012-08-24,659.51,669.48,655.55,663.22,15619300,663.22
2012-08-23,666.11,669.90,661.15,662.63,15004600,662.63
```

7. Thank you, Professor. Hey, everyone, this is Davia. So here's an example of what the data looks like that Professor was talking about. This is basically a comma separated file, as you can see. And as you observe that the CSV is in the reverse order (意思是 CSV 文件中的 date 是由大到小的, 而我們想讓時間由小到大), but soon we will teach you how to fix that.

```
1  import pandas as pd
2
3
4  def test_run():
5      df = pd.read_csv("data/AAPL.csv")
6      print df #print entire dataframe
7
8
9  if __name__ == "__main__":
10     test_run()
```

if __name__ == "__main__"之作用見本文件最後

8. Pandas provide a number of functions that makes it easy to read in data like the .csv file we just had a look at. Here's a code that reads in AAPL.csv into a data frame. So, first of all we will have to import the pandas library. To avoid writing pandas every time we use a functionality of it, we rename it as pd. So, this is the main function, which will call the test run function. Let's have a look what's there in it. pd.read_csv, as the name suggests, will read AAPL.csv into a data frame, which we name it as df. As of now, imagine dataframe as a structure similar to the 2D arracy. That is, with rows and columns.

	Date	Open	High	Low	Close	Volume	Adj Close
0	2012-09-12	666.85	669.90	656.00	669.79	25410600	669.79
1	2012-09-11	665.11	670.10	656.50	660.59	17987400	660.59
2	2012-09-10	680.45	683.29	662.10	662.74	17428500	662.74
3	2012-09-07	678.05	682.48	675.77	680.44	11773800	680.44
4	2012-09-06	673.17	678.29	670.80	676.27	13971300	676.27
5	2012-09-05	675.57	676.35	669.60	670.23	12013400	670.23
6	2012-09-04	665.76	675.14	664.50	674.97	13139000	674.97
7	2012-08-31	667.25	668.60	657.25	665.24	12082900	665.24
8	2012-08-30	670.64	671.55	662.85	663.87	10810700	663.87
9	2012-08-29	675.25	677.67	672.60	673.47	7243100	673.47
10	2012-08-28	674.98	676.10	670.67	674.80	9550600	674.80
11	2012-08-27	679.99	680.87	673.54	675.68	15250300	675.68
12	2012-08-24	659.51	669.48	655.55	663.22	15619300	663.22

Let's go ahead and print this. So here's the entire csv file on your console. As you can see, the entire data is loaded in your console,

```
1  import pandas as pd
2
3
4  def test_run():
5      df = pd.read_csv("data/AAPL.csv")
6      print df.head() #print entire dataframe
7
8
9  if __name__ == "__main__":
10     test_run()
```

but just to have an idea of the .csv file, you can just print the top five rows of the data frame. This is how you do it. Data frame dot head. Dot head is the functionality provided by the pandas for the data frame that would help you to view just the top five lines of the csv. That will give you a rough idea of

what the .csv actually contains.

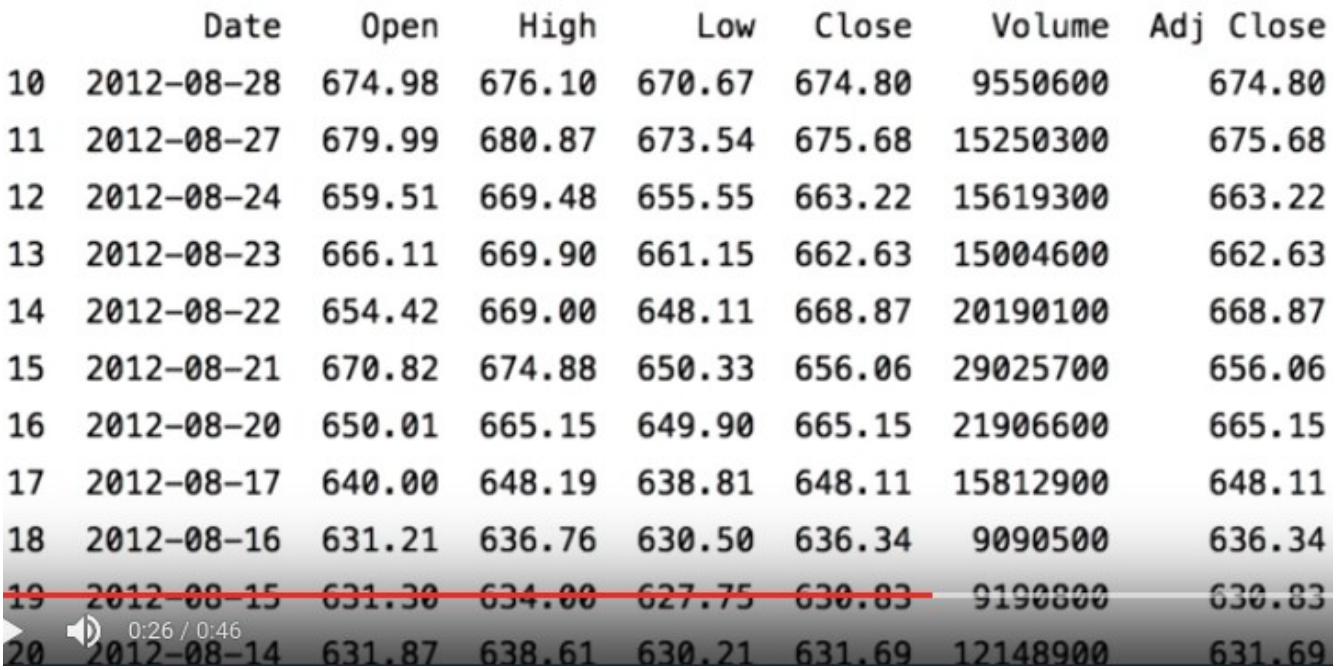
	Date	Open	High	Low	Close	Volume	Adj Close
0	2012-09-12	666.85	669.90	656.00	669.79	25410600	669.79
1	2012-09-11	665.11	670.10	656.50	660.59	17987400	660.59
2	2012-09-10	680.45	683.29	662.10	662.74	17428500	662.74
3	2012-09-07	678.05	682.48	675.77	680.44	11773800	680.44
4	2012-09-06	673.17	678.29	670.80	676.27	13971300	676.27

Let's go ahead and print this. So here it is. Just the top five lines of your data frame. You can observe that all the columns of the .csv can be viewed here. You will also observe there is a column that is not named and has values 0, 1, 2, 3. And this is not from the .csv. These are called index for the data frame, which help you to access rows. Similarly, you can view last five values using the df.tail.

```
1  import pandas as pd
2
3
4  def test_run():
5      df = pd.read_csv("data/AAPL.csv")
6      print df[10:21]#rows between index 10 and 20
7
8
9  if __name__ == "__main__":
10     test_run()
```

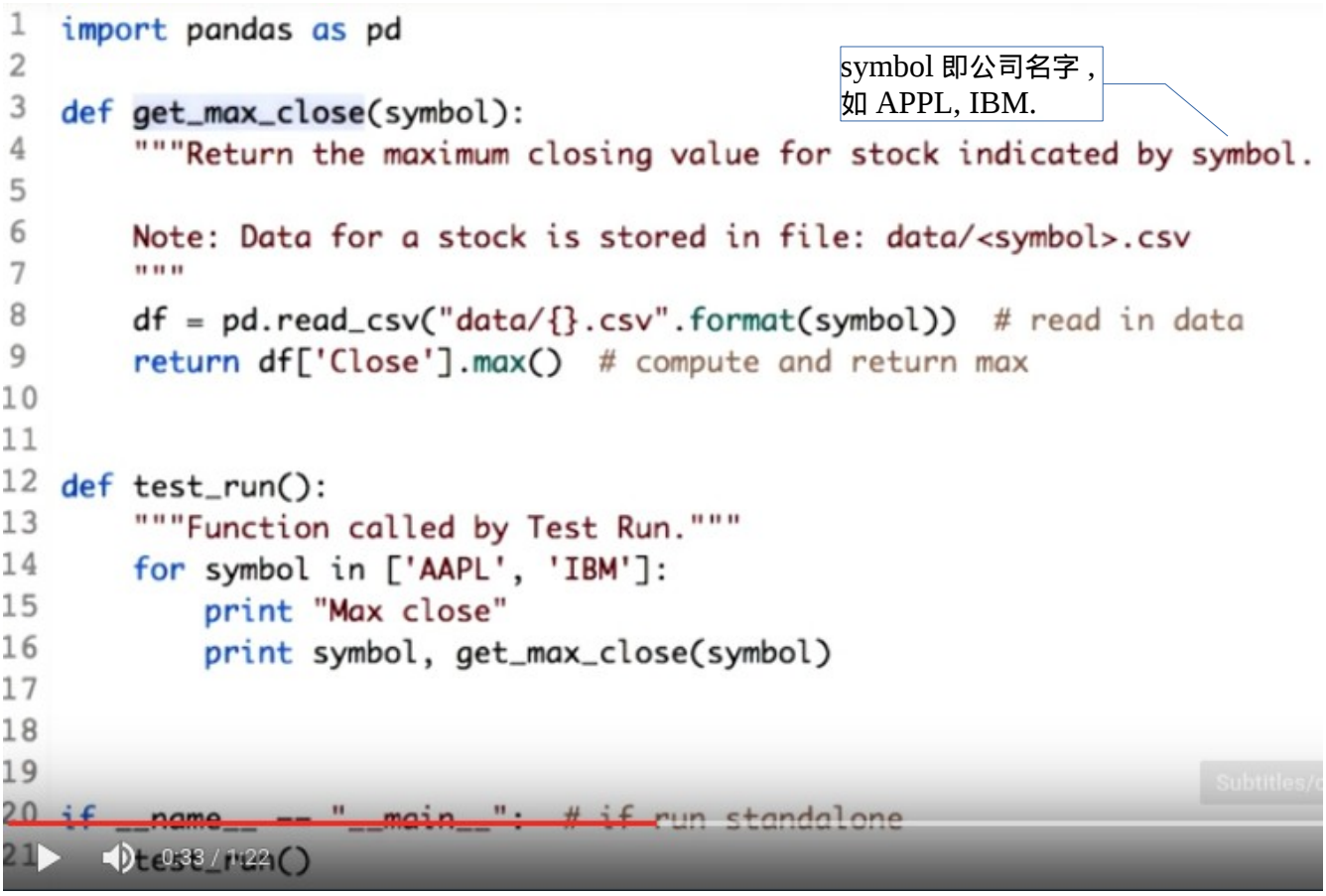
9. So now let's do some interesting stuff. What if I want to view rows from the DataFrame in between some random values and not the head and the tail? We can do something of this kind. If you want data from index, 10 to 20, just add this line.

print df[10:21]



	Date	Open	High	Low	Close	Volume	Adj Close
10	2012-08-28	674.98	676.10	670.67	674.80	9550600	674.80
11	2012-08-27	679.99	680.87	673.54	675.68	15250300	675.68
12	2012-08-24	659.51	669.48	655.55	663.22	15619300	663.22
13	2012-08-23	666.11	669.90	661.15	662.63	15004600	662.63
14	2012-08-22	654.42	669.00	648.11	668.87	20190100	668.87
15	2012-08-21	670.82	674.88	650.33	656.06	29025700	656.06
16	2012-08-20	650.01	665.15	649.90	665.15	21906600	665.15
17	2012-08-17	640.00	648.19	638.81	648.11	15812900	648.11
18	2012-08-16	631.21	636.76	630.50	636.34	9090500	636.34
19	2012-08-15	631.30	634.00	627.75	630.83	9190800	630.83
20	2012-08-14	631.87	638.61	630.21	631.69	12148900	631.69

Let's see the output. Here it is. All the data between the index 10 and 20 are displayed. But you might also observe that if you want data between 10 and 20, you have to mention 10:21. Because 21 is not inclusive in the range. This operation is called slicing and it is a very important operation in Python pandas, which you will encounter in the future lesson.




```
Max close
AAPL 680.44
Max close
IBM 209.5
```

選行: df[10:21]
選列: df['Close']

10. Now let's do some more processing on the data frame. We can start with finding the maximum closing value for each of the stock AAPL and IBM. So here's the code. The test_run function simply loops over two symbols, AAPL and IBM, and will print the maximum closing value of each of the stock. Let's call the function get_max_close along with the symbol. Here's the function that will compute the maximum closing value. Let's see what get_max_close function does. The first step would be to read in the csv into the data frame. The next step would be to get only the closing values from the entire data frame, which means we have to extract the column close. This is how you do it. df[, pass the parameter of the column name, that is, 'Close'. Make sure you include the inverted commas. The last step is to calculate the maximum value, and it is as simple as calling the .max()

function over the extracted data. Let's go ahead and print this. Here is your output. The max close for the AAPL is 680.44 and the max close for the IBM is 209.5.

11. Your task is to calculate the mean volume for each of the given symbols. You can start with extracting the volume column form the data frame and then finding the mean. I'll come back with the solution. Good luck.

```
1  import pandas as pd
2
3
4  def get_mean_volume(symbol):
5      """Return the mean volume value for stock indicated by symbol.
6
7      Note: Data for a stock is stored in file: data/<symbol>.csv
8      """
9      df = pd.read_csv("data/{}.csv".format(symbol))  # read in data
10     return df['Volume'].mean()  # compute and return mean
11                                 只有這句跟前面不同
12
13 def test_run():
14     """Function called by Test Run."""
15     for symbol in ['AAPL', 'IBM']:
16         print "Mean Volume"
17         print symbol, get_mean_volume(symbol)
18
19
20 if __name__ == "__main__":  # if run standalone
21     test_run()
```

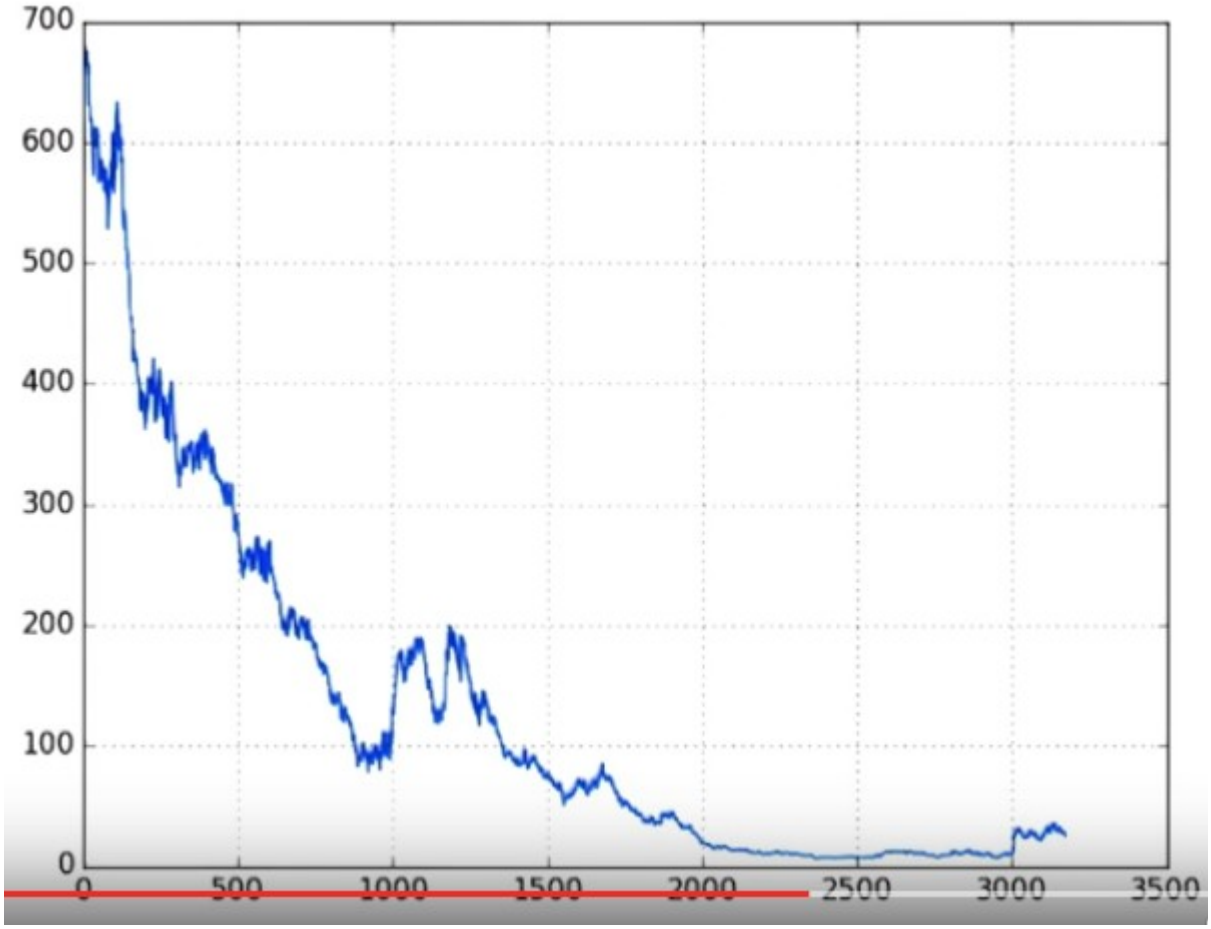
0:09 / 0:24

```
Mean Volume
AAPL 21491431.3386
Mean Volume
IBM 7103570.80315
```

12. So here's the solution to find the mean volume for the given stroke. As I explained, the first step would be to extract the volume column from the data frame. The next step is to find the mean. It can be done by calling the mean function. Let's run this code. Here you go. The mean value for Apple, and the mean value for IBM. I hope you enjoyed the quiz.

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  def test_run():
5      df = pd.read_csv("data/AAPL.csv")
6      print df['Adj Close']
7      df['Adj Close'].plot()
8      plt.show()  # must be called to show plots
9
10
11 if __name__ == "__main__":
12     test_run()
```

13. Now let's do some plotting. It's easy to plot data in the data frame. Here's how you plot Apple's adjusted close. First let's call a library that would help us to do this. We import a library name, matplotlib. Do not worry about the details. You will learn them eventually in the further lessons. But to plot the adjusting close, we first need the adjusting close data from the data frame. And, as you learned in the previous video, we can slice over the column using the square brackets. Plotting the adjusting close is as simple as calling a plot function. To show the plot on your screen, we need to add one more line and this plot.show.



Now let's run this code. Here's your first graph. You can observe there is no x-axis label, no y-axis label, no header also the data is printed in reverse order since the CSV is in the reverse order. So the Apple prices are not moving down, they are just printed inversely. In the coming lessons, you will learn how to fix it. As of now, enjoy the power of the Python Pandas that can plot information using just one line of code. Get ready to plot some data by yourself. I'll be back with a quiz.

14. So here's the question for you. Plot the high prices for the IBM. You can approach this problem by first getting the CSV data of the IBM followed by getting the high prices from the data frame and then plotting it. You can refer to the previous example. Good luck.

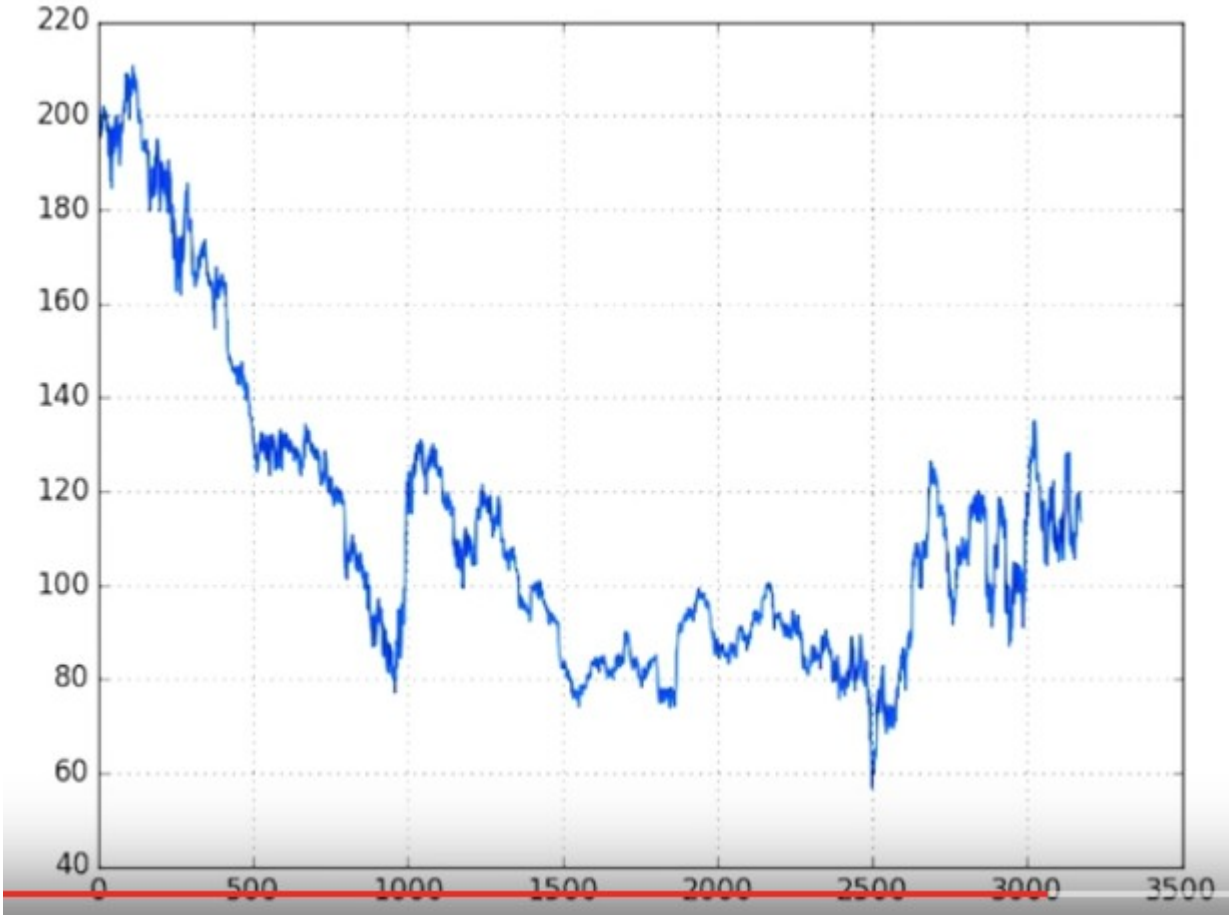
```python
import pandas as pd
import matplotlib.pyplot as plt

def test_run():
    df = pd.read_csv("data/IBM.csv")
    ''' TO-DO
    Plot "High" prices for "IBM"
    '''
    print df['High']
    df['High'].plot()
    plt.show()  # must be called to show plots


if __name__ == "__main__":
    test_run()
```

只有這句跟前面不同

15. Here's the solution. You can get the csv of the IBM by using IBM.csv. The next step was the get the high prices from the data frame. df[high] will do that for you.



And finally, you go ahead and plot it. Let's see how the graph looks like. I hope you got a graph similar to this. An advice I would like to give you at this point is, you will get hold of Python easily if you experiment. Try different options and see what works and what does not. During the course you will realize why some things worked and why just some things failed.

```python
import pandas as pd
import matplotlib.pyplot as plt

def test_run():
    df = pd.read_csv("data/AAPl.csv")
    df[['Close','Adj Close']].plot()
    plt.show()  # must be called to show plots


if __name__ == "__main__":
    test_run()
```

只有這句跟前面不同

16. I'll sign off by showing you some pair of pandas. We are about to plot two columns simultaneously on one graph. That is close and adjusted close for the Apple stock. Don't worry about how to extract multiple columns from the data frame. But intuitively you use a double square brackets and pass the two column names, that is 'Close' and 'Adj Close'.

We go ahead and plot this. Here's the graph. You can observe two lines. One is blue, which corresponds to Close; and one is green, which corresponds to Adj Close. Observe that we did not write code to print the legend, or give color to each of the graph lines. This is the pair of the Python pandas. The blue line corresponds to the close value, and the green line corresponds to the adjusted close values. You will learn in the further lesson why there is a difference. That's all for now, I'll see you in the next lesson. Happy coding.

From online:

What does if __name__ == "__main__": do?

When your script is run by passing it as a command to the Python interpreter,

python myscript.py

all of the code that is at indentation level 0 gets executed. Functions and classes that are defined are, well, defined, but none of their code gets ran. Unlike other languages, there's no main() function that gets run automatically - the main() function is implicitly all the code at the top level.

In this case, the top-level code is an if block. __name__ is a built-in variable which evaluate to the name of the current module. However, if a module is being run directly (as in myscript.py above), then __name__ instead is set to the string "__main__". Thus, you can test whether your script is being run directly or being imported by something else by testing

```
if __name__ == "__main__":
    ...
```

If that code is being imported into another module, the various function and class definitions will be imported, but the main() code (即指 if A else B 中的 A) won't get run. As a basic example, consider the following two scripts:

```
# file one.py
def func():
    print("func() in one.py")

print("top-level in one.py")

if __name__ == "__main__":
    print("one.py is being run directly")
else:
    print("one.py is being imported into another module")

# file two.py
import one

print("top-level in two.py")
one.func()

if __name__ == "__main__":
    print("two.py is being run directly")
```

```
else:
    print("two.py is being imported into another module")
```

Now, if you invoke the interpreter as

```
python one.py
```

The output will be

```
top-level in one.py
one.py is being run directly
```

If you run two.py instead:

```
python two.py
```

You get

```
top-level in one.py
one.py is being imported into another module
top-level in two.py
func() in one.py
two.py is being run directly
```

Thus, when module one gets loaded, its __name__ equals "one" instead of __main__.