Python Introduction

Kevin Sheppard
University of Oxford
www.kevinsheppard.com

September 2019

Contents

Installing

Install Anaconda

1. Download the Anaconda Python/R Distribution 2019.07 (or later).
2. When the download is complete, install into your user account.

Install Visual Studio Code and the Python extension

1. Download VS Code and install
2. Install the Python extension by clicking on Extensions and searching for "Python"
3. Open the `mfe-introduction` folder created in the previous step
4. Create a file called `second.py` and enter

```
#%%

print("Python may be harder to learn than other languages since")
print("there is rarely a single approach to completing a task.")
```

5. Click on Run Cell

Note the `#%%` makes it a magic cell

Install Pycharm Professional

1. Download PyCharm Professional and install using the 30-day trial. You can get a free copy using your academic email address if you want to continue after the first 30 days.
2. Open PyCharm, and create a new project called `mfe-introduction`
3. Open File > Setting and select Python Interpreter. Select the Anaconda interpreter if it is not already selected.
4. Create a new python file called `first.py` and enter

```
print("Python has a steeper curve than MATLAB but more long-run upside")
```

5. Right-click on this file, and select “Run”.

Lesson 1

Getting Started

This lesson covers:

- Opening a terminal window
- Launching Jupyter notebook
- Running IPython in a Terminal
- Running IPython in Jupyter QtConsole
- Executing a standalone Python file in IPython
- Optional
  - Jupyter notebooks in VSCode
  - Jupyter notebooks in PyCharm Professional

1.1 Opening an Anaconda Terminal

An Anaconda terminal allows `python` to be run directly. It also allows other useful programs, for example `pip`, the Python package manager to be used to install packages that are not available through Anaconda.

Windows

Launch Anaconda Prompt from the start menu.

OSX and Linux

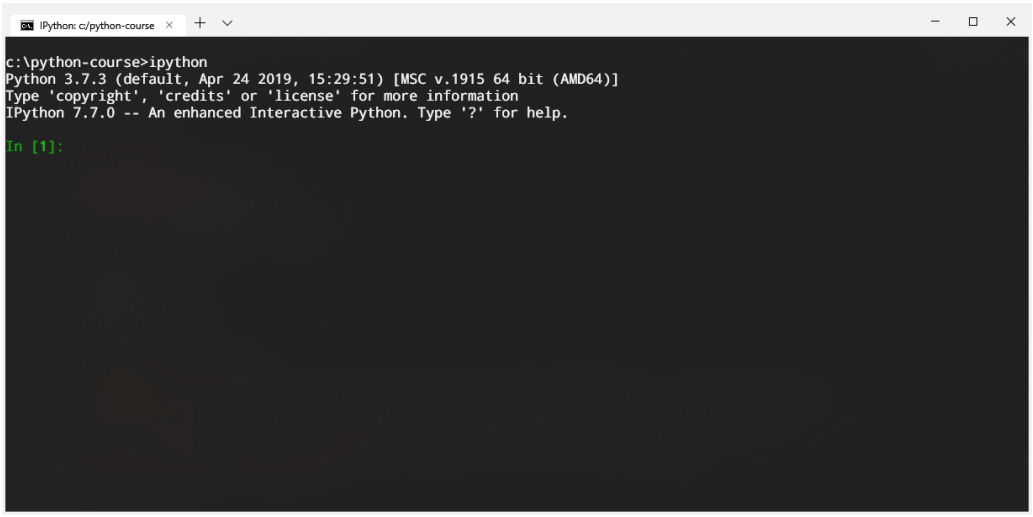
Open the terminal (instructions depend on your distribution). If you allowed conda to initialize, then you should be ready to call Anaconda's python and supporting functions. If not, you should

```
cd ~/anaconda3/bin
./conda init
```

and then reopen your terminal.

1.2 Running IPython in a Terminal

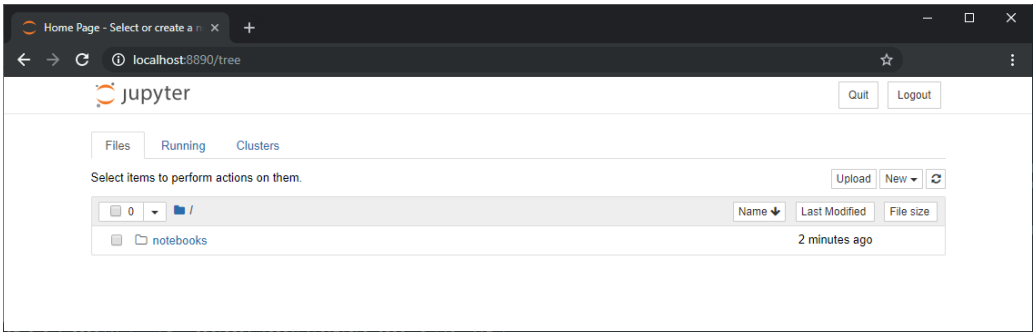
1. Open a terminal.
2. Run IPython by entering `ipython` in the terminal window. You should see a window like the one below with the iconic `In [1]` indicating that you are at the start of a new IPython session.



IPython in Windows Terminal

1.3 Launching Jupyter notebook

1. Launch Jupyter Notebook from the Start Menu or launcher.
2. Change directory to the location where you store your notebooks.



Jupyter Notebook

1.4 Executing a standalone Python file in IPython

1. Open a text editor and enter the following lines. Save the file as lesson-2.py. Note that Python is white-space sensitive, and so these lines should **not** not indented.

```
from math import exp, log

x = exp(1)
y = log(x)

print(f'exp(1)={x}, log(exp(1))={y}')
```

2. Run the code in an IPython session using `%run -i lesson-2.py`. Note: you should create the python file in the same directory as the notebook.

If everything works as expected, you should see

```
exp(1)=2.718281828459045, log(exp(1))=1.0
```

1.5 Jupyter notebooks in VSCode

Visual Studio Code (or VS Code) is a lightweight IDE that supports adding features through extensions. The key extension for working with notebooks is Python extension for Visual Studio Code. With this extension installed, it is possible to use a special file format called Magic Python to write notebook-like files that can be exported to Jupyter notebook files.

1. Install VS Code and the Python extension
2. Open the command palette and enter "Jupyter start" and select the only available item.
3. This is a Python file that supports a cell demarcation using `#%%` for code cells and `#%% [markdown]` for cells that contain markdown code. Note that markdown text **must** be either:

    - Surrounded by triple quotes, e.g. `"""markdown text"""` or `'''markdown text'''`; e.g.,

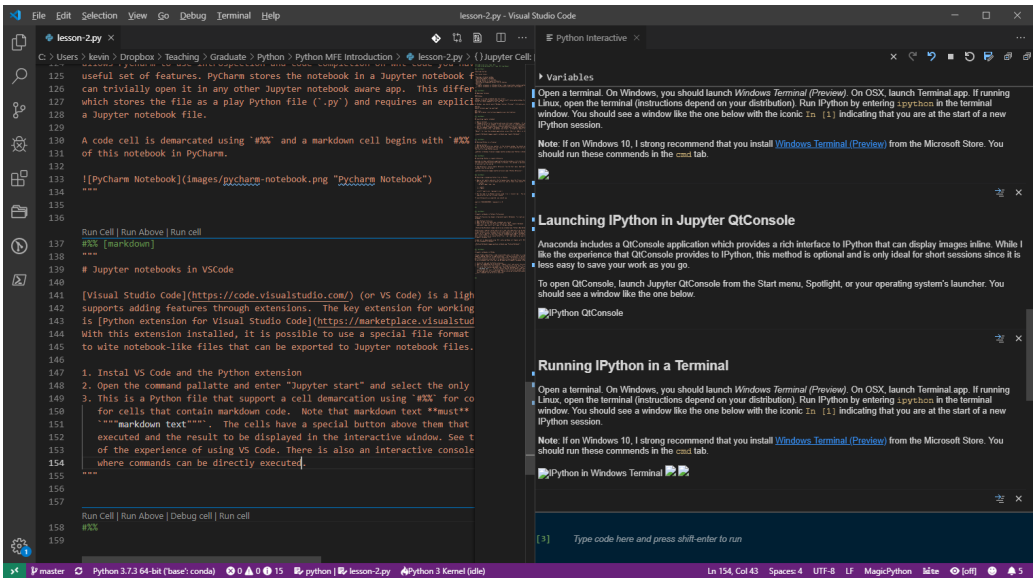
    ```
    """
    # Cell Heading

    Likeness darkness. That give brought creeping. Doesn't may. Fruit kind
    midst seed. Creature, let under created void god to. Them day was Was
    creature set it from. Fourth. Created don't man. Man. Light fourth
    light given the he image first multiply after deep she'd great. Morning
    likeness very have give also fowl third land beast from moving thing
    creepeth herb creeping won't fifth. Us bring was our beast wherein our
    void and green he fruit kind upon a given, saying fruit, moveth face
    forth. His you it. Good beginning hath.
    """
    ```

    - Or commented # (with a single space) at the start of each line,

```
# # Cell Heading
#
# Likeness darkness. That give brought creeping. Doesn't may. Fruit kind
# midst seed. Creature, let under created void god to. Them day was Was
# creature set it from. Fourth. Created don't man. Man. Light fourth
# light given the he image first multiply after deep she'd great. Morning
# likeness very have give also fowl third land beast from moving thing
# creepeth herb creeping won't fifth. Us bring was our beast wherein our
# void and green he fruit kind upon a given, saying fruit, moveth face
# forth. His you it. Good beginning hath.
```

The cells have a special button above them that allows the contents to be executed and the result to be displayed in the interactive window. See the screenshot below for an example of the experience of using VS Code. There is also an interactive console at the bottom left where commands can be directly executed.

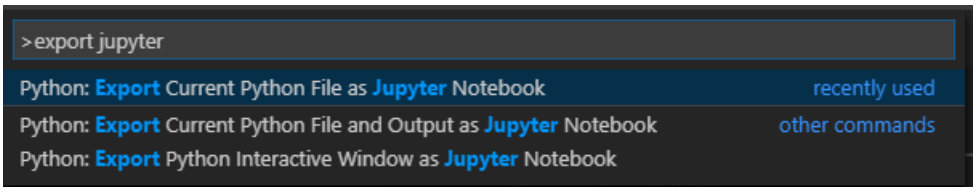


VS Code Notebook

Importing an exiting notebook in VS Code

VS Code only understands Magic Python files as notebook-like documents, and so `.ipynb` files must be converted to use. The process of importing is simple:

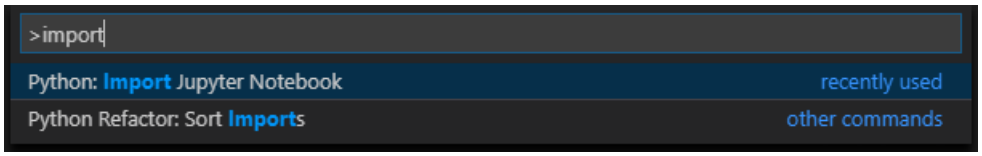
1. Open a Jupyter notebook file
2. Click on Import in the popup that appears.



VS Code Export

Exporting to an Jupyter notebook

To export a Magic Python file, open the command palette and enter "import jupyter". Select the option to import the notebook.



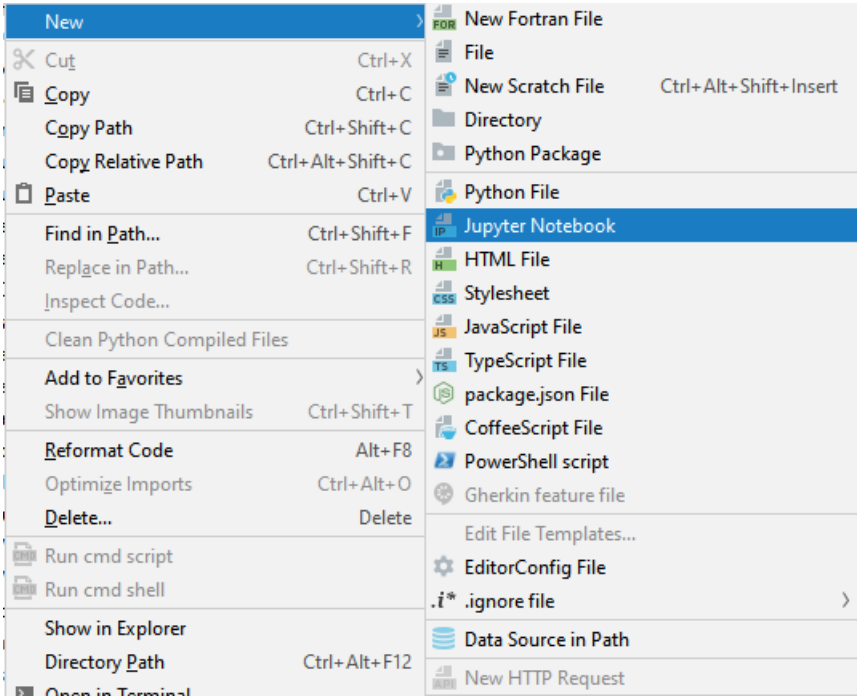
VS Code Import

1.6 Jupyter notebooks in PyCharm Professional

PyCharm Professional is my recommended approach if you are going to use Python throughout the course. It provides the best experience and can be acquired for free using the student program.

PyCharm Professional has deeply integrated Jupyter Notebooks. To create an IPython notebook:

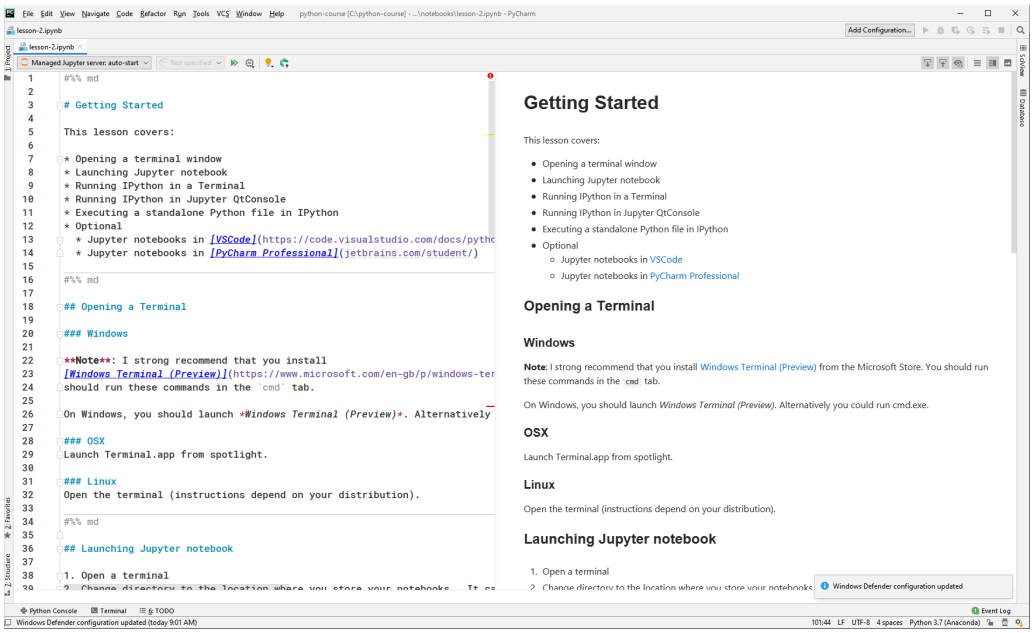
1. Open PyCharm Profession
2. Open the directory where your notebooks are stored
3. Right-click on the root directory and select `New > Jupyter Notebook`. Give your file a meaningful name, and it will open in the main window.



PyCharm New Notebook

PyCharm uses a special syntax where cells look like code and so can be edited like text. This allows PyCharm to use introspection and code completion on the code you have written, a highly useful set of features. PyCharm stores the notebook in a Jupyter notebook file (`.ipynb`), which means that you can trivially open it in any other Jupyter notebook aware app. This differs from Section 1.5 which stores the file as a play Python file (`.py`) and requires an explicit export to a Jupyter notebook file.

A code cell is demarcated using `#%%` and a markdown cell begins with `#%% md`. Below is a screenshot of this notebook in PyCharm.



PyCharm Notebook

Magic Python in PyCharm

PyCharm supports Magic Python cell execution. To use Magic Python, you need to enable *Scientific Mode* in the View menu. You can then use `#%%` to indicate the start and end of cells. Individual Cells can be executed in the console by pressing CTRL+Enter.

1. In PyCharm, right-click on the root directory and select `New > Python File`. Give your file a meaningful name.
2. Enter

```
#%%
print('This is the first cell')


#%%
print('This is not executed when the first cell is run')
```

3. Enable Scientific Mode in the View menu.
4. Run the first cell by placing you mouse in the cell and pressing CTRL+Enter.
5. Run the second cell by clicking on the Play button (arrow) that appears in the gutter of the editor.

Note: Magic Python in PyCharm only supports python code, and so it is not possible to mix Markdown text and Python in the same file.

Lesson 2

Basic Python Types

This lesson covers:

- Inputting scalars and strings
- Lists
- Dictionaries

Problem: Input scalar floating point and integers

1. Create a variable called `scalar_float` containing π to 4 digits.
2. Create a variable called `scalar_int` containing 31415.
3. Print each value using the `print` function.

Problem: Create a string and an f-string

1. Create a variable called `a_string` containing `This is a string`
2. Create a f-string the prints `The value of scalar_float is 3.1415` using the variable created in the previous step
3. Create two string, `first` containing `String concatenation` and the second containing `is like addition`, and join the two using `+` to produce `String concatenation is like addition`.

Problem: Create a list

1. Create a list containing `scalar_float` and `scalar_int`
2. Add `a_string` to the list.
3. Select the the lst two elements of the list

Problem: Create a list of lists

1. Create a list containing the two lists `[1, 2, 3]` and `[4, 5, 6]`
2. Select the element 5 from the nested list

Problem: Create a dictionary

1. Create a dictionary containing the key-value pairs `"float"` and 3.1415, `"int"` and 31415, and `"string"` and `"three-point-one-four-one-five"`.

Problem: Lookup and Change a value

1. Look up the value of `"float"`.
2. Change the value of `"float"` to `22 / 7`.

Problem: Add and remove a key

1. Add the new key “better_float” with the value 3.141592.
2. Remove the key “float” and its value.

Lesson 3

Importing Modules

This lesson covers:

- Module import

Problem: Importing Modules

Python is a general-purpose programming language and is not specialized for numerical or statistical computation. The core modules that enable Python to store and access data efficiently and that provide statistical algorithms are located in modules. The most important are:

- NumPy (`numpy`) - provide the basic array block used throughout numerical Python
- pandas (`pandas`) - provides DataFrames which are used to store data in an easy-to-use format
- SciPy (`scipy`) - Basic statistics and random number generators. The most important submodule is `scipy.stats`
- matplotlib (`matplotlib`) - graphics. The most important submodule is `matplotlib.pyplot`.
- statsmodels (`statsmodels`) - statistical models such as OLS. The most important submodules are `statsmodels.api` and `statsmodels.tsa.api`.

Begin by importing the important modules.

Problem: Canonical Names

Use the `as` keyword to import the modules using their canonical names:

Module	Canonical Name
numpy	np
pandas	pd
scipy	sp
scipy.stats	stats
matplotlib.pyplot	plt
statsmodels.api	sm
statsmodels.tsa.api	tsa

Import the core modules using `import` *module* `as` *canonical*.

Problem: Importing individual functions

1. Import `array`, `sqrt`, `log` and `exp` from NumPy.
2. Import `OLS` from `statsmodels.regression.linear_model`
3. Import the `stats` module from `scipy`

Lesson 4

Series and DataFrames

This lesson covers:

- Manually inputting data in scalars, vectors, and matrices
- Basic mathematical operations
- Saving and loading data

Data September 2018 prices (adjusted closing prices) for the S&P 500 EFT (SPY), Apple (AAPL) and Google (GOOG) are listed below:

Date	SPY Price	AAPL Price	GOOG Price
Sept4	289.81	228.36	1197.00
Sept5	289.03	226.87	1186.48
Sept6	288.16	223.10	1171.44
Sept7	287.60	221.30	1164.83
Sept10	288.10	218.33	1164.64
Sept11	289.05	223.85	1177.36
Sept12	289.12	221.07	1162.82
Sept13	290.83	226.41	1175.33
Sept14	290.88	223.84	1172.53
Sept17	289.34	217.88	1156.05
Sept18	290.91	218.24	1161.22
Sept19	291.44	216.64	1158.78

Prices in September 2018

Problem: Input scalar data

Create 3 variables, one labeled `spy`, one labeled `aapl` and one labeled `goog` that contain the September 4 price of the asset. For example, to enter the Google data

```
goog = 1197.00
```

Problem: Print the values

Print the values of the three variables you created in the previous step using `print`.

Problem: Print the values with formatting

Print the values of the three variables you created in the previous step using format strings following the pattern TICKER: Value. For example, you can print the value of Google using `print(f'GOOG: {goog}')`.

Problem: Input a Vector

Create vectors for each of the days in the Section **??** named `sep_xx` where `xx` is the numeric date. For example,

```
import pandas as pd

sep_04 = pd.Series([289.81,228.36,1197.00], index=['SPY','AAPL','GOOG']);
```

Problem: Create a Vector of Dates

Use the pandas function `pd.to_datetime` to convert a list of string dates to a pandas `DateTimeIndex`, which can be used to set dates in other arrays. For example, the first two dates are

```
dates_2 = pd.to_datetime(['4-9-2018','5-9-2018'])
print(dates_2)
```

which produces

```
DatetimeIndex(['2018-04-09', '2018-05-09'], dtype='datetime64[ns]', freq=None)
```

Create a vector containing all of the dates in the table.

Problem: Input a Vector with Dates

Create vectors for each of the ticker symbols in Section **??** named spy, aapl and goog, respectively. Use the variable `dates` that you created in the previous step.

For example

```
goog = pd.Series([1197.00,1186.48,1171.44,...], index=dates)
```

Problem: Create a DataFrame

Create a DataFrame named `prices` containing Section **??**. Set the column names equal to the ticker and set the index to the dates you created previously.

```
prices = pd.DataFrame([[289.81, 228.36, 1197.00], [289.03, 226.87, 1186.48]],
                      columns = ['SPY', 'AAPL', 'GOOG'],index=dates_2)
```

Problem: Construct a DataFrame from Series

Create a second DataFrame named prices_row from the row vectors previously entered such that the results are identical to prices. For example, the first two days worth of data are

```
pricess_row = pd.DataFrame([Sep04, Sep05])
# Set the index after using concat to join
pricess_row.index = dates_2
```

Create a third DataFrame named prices_col from the 3 column vectors entered such that the results are identical to prices

```
prices_col = pd.DataFrame([SPY,APPL,GOOG]).T
```

Note: The `.T` above transposes the 2-d array since `DataFrame` builds the array by rows.

Verify that all three matrices are identical by printing the difference, e.g.,

```
print(pricescol - prices)
```

and that all elements are 0.

Save the price data

This block saves prices to a HDF file for use in later lessons. The function used to save the data is covered in a later lesson.

```
# Setup: Save prices, goog and sep_04 into a single file for use in other lessons

# Only run if prices has been defined
if 'prices' in globals():
    with pd.HDFStore('data/data.h5', mode='w') as h5:
        h5.put('prices', prices)
        h5.put('goog', goog)
        h5.put('sep_04', sep_04)
```

Lesson 5

Calling Functions

This lesson covers:

- Calling functions with more than one input and output
- Calling functions when some inputs are not used

Read the data in momentum.csv and creating some variable. This cell uses some magic to automate repeated typing.

```
# Setup: Load the momentum data
import pandas as pd

momentum = pd.read_csv('data/momentum.csv')

print(momentum.head())

mom_01 = momentum['mom_01']
mom_10 = momentum['mom_10']
```

This data set contains 2 years of data on the 10 momentum portfolios from 2016–2018. The variables are named mom_XX where XX ranges from 01 (work return over the past 12 months) to 10 (best return over the past 12 months).

Problem: Calling Functions

Functions were used in the previous lesson. Get used to calling functions by computing the mean, std, kurtosis, max, and min of the 10 momentum portfolios. Also, explore the help available for calling functions `?` operator. For example,

```
momentum.std?
```

opens a help window that shows the inputs and output, while

```
help(momentum.std)
```

shows the help.

Use the functions `mean`, `std`, `skew` and `kurt` to print the values for `mom_01`.

Problem: Use NumPy and SciPy functions

Use the NumPy functions `mean` and `std` and the SciPy `stats` functions `skew` and `kurtosis` to produce the same output.

Problem: Calling Functions with 2 Outputs

Some useful functions return 2 or more outputs. One example is `np.linalg.slogdet` computes the signed log determinant of a square array. It returns two output, the sign and the log of the absolute determinant.

Use this function to compute the sign and log determinant of the 2 by 2 array:

```
1  2
2  9
```

Problem: Calling Functions with 2 Inputs

Many functions take two or more inputs. Like outputs, the inputs are simply listed in order separated by commas. Use `np.linspace` to produce a series of 11 points evenly spaced between 0 and 1.

Problem: Calling Functions using Keyword Arguments

Many functions have optional arguments. You can see these in a docstring since optional arguments take the form `variable=default`. For example, see the help for `np.mean`

which is

```
pd.DataFrame.std(self, axis=None, skipna=None, level=None, ddof=1, numeric_only=None)
```

`std` computes the standard deviation.

This tells us that only `self` (which is the DataFrame) is required and that the other 5 inputs can be omitted if you are happy with the defaults. However, if we want to change some of the optional inputs, then we can directly use the inputs name in the function call.

By default `std` divides by `n-1`. The `1` can be set using `ddof`.

Compute `std` using `ddof=0` on the momentum data.

Lesson 6

Custom Functions

This lesson covers:

- Writing a custom function

Problem: Writing a Custom Function

Custom functions will play an important role later in the course when estimating parameters. Construct a custom function that takes two arguments, mu and sigma2 and computes the likelihood function of a normal random variable.

$$f(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Use `def` to start the function and compute the likelihood of:

$$x = 0, \mu = 0, \sigma^2 = 1.$$

The text in the triple quotes is the docstring which is optional.

Exercises

Exercise: Custom Function

Write a function named summary_stats that will take a single input, x, a DataFrame and return a DataFrame with 4 columns and as many rows as there were columns in the original data where the columns contain the mean, standard deviation, skewness and kurtosis of x.

Exercise: Custom Function

Change your previous function to return 4 outputs, each a pandas Series for the mean, standard deviation, skewness, and the kurtosis.

Returning multiple outputs uses the syntax

```
return w, x, y, z
```

Lesson 7

Using DataFrames

This lesson introduces:

- Basic mathematical operations on DataFrames
- Common DataFrame methods (functions)
- Computing percentage change (returns)

Problem: Addition and Subtraction

Add the prices of the three series together using `.sum(axis=1)`. Add the prices in `sep_04` to the prices of `goog`. What happens?

```
# Setup: Load prices
import pandas as pd
prices = pd.read_hdf('data/data.h5', 'prices')
sep_04 = pd.read_hdf('data/data.h5', 'sep_04')
goog = pd.read_hdf('data/data.h5', 'goog')
```

Problem: Multiplication

Multiply the price of Google by 2.

Problem: Constructing portfolio returns

Set up a vector or portfolio weights $w = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ and compute the price of a portfolio with $\frac{1}{3}$ share of each.

Note: Division uses the slash operator (/).

Problem: Compute Returns

Compute returns using

```
returns = prices.pct_change()
```

which computes the percentage change.
Additionally, extract returns for each name using

```
spy_returns = returns['SPY']
```

Problem: Compute Log Returns

```
import numpy as np

log_returns = np.log(prices).diff()
```

first difference of the natural log of the prices. Mathematically this is $r_t = \ln(P_t) - \ln(P_{t-1}) = \ln\left(\frac{P_t}{P_{t-1}}\right) \approx \frac{P_t}{P_{t-1}} - 1$.

Problem: Mean, Standard Deviation and Correlation

Using the function mean, compute the mean of the three returns series one at a time. For example

```
goog_mean = goog_returns.mean()
```

Next, compute the mean of the matrix of returns using

```
retmean = returns.mean()
```

What is the relationship between these two? Repeat this exercise for the standard deviation (`std()`). Finally, compute the correlation of the matrix of returns (`corr()`).

Problem: Summing all elements

Compute the sum of the columns of returns using `.sum()`. How is this related to the mean computed in the previous step?

Problem: Maximum and Minimum Values

Compute the minimum and maximum values of the columns of returns using the `min()` and `max()` commands.

Problem: Rounding Up, Down and to the Closest Integer

Rounding up is handled by ceil, rounding down is handled by floor and rounding to the closest integer is handled by round. Try all of these commands on 100 times returns. For example,

```
rounded = (100*returns).round()
```

Use `ceil` and `floor` to round up and down, respectively.

Problem: Element-by-Element Multiplication

Mathematical commands in Python are element-by-element, except the `@` operator which is matrix multiplication and uses the rules of linear algebra.

Multiply the returns of Google and SPY together using the dot operator.

Lesson 8

Accessing Elements in DataFrames

This lesson covers:

- Assessing specific elements in Pandas Series and DataFrames

Accessing elements in an array or a DataFrame is a common task. To begin this lesson, clear the workspace set up some vectors and a 5×5 array. These vectors and matrix will make it easy to determine which elements are selected by a command.

Problem: Numeric indexing Series and DataFrame

Repeat the previous questions on `y_s` and `x_df` using `.iloc`.

Problem: Selecting by Name in Series and DataFrames

Using `x_name` and `y_name`:

1. Select the (0,2) and the (2,0) element of `x_name`.
2. Select the 2nd row of `x_name` using `.loc`.
3. Select the 2nd columns of `x_name` using `.loc`.
4. Select the 2nd element of `y_name` using both `[]` and `loc`.
5. Select the 2nd and 4th rows and 1st and 3rd columns of `x_name`.

Problem: Selecting Data by Date

Load the data in momentum.csv.

```
# Setup: Load the momentum data

import pandas as pd

momentum = pd.read_csv('data/momentum.csv', index_col='date', parse_dates=True)
momentum.head()
```

1. Select returns on February 16, 2016.
2. Select return in March 2016.
3. Select returns between May 1, 2016, and June 15, 2016

Lesson 9

Accessing Elements in NumPy Arrays

This lesson covers:

- Accessing specific elements in NumPy arrays

Accessing elements in an array or a DataFrame is a common task. To begin this lesson, clear the workspace set up some vectors and a 5×5 array. These vectors and matrix will make it easy to determine which elements are selected by a command.

Using `arange` and `reshape` to create 3 arrays:

- 5-by-5 array `x` containing the values 0,1,...,24
- 5-element, 1-dimensional array `y` containing 0,1,...,4
- 5-by-1 array `z` containing 0,1,...,4

9.1 Zero-based indexing

Python indexing is 0 based so that the first element has position `0`, the second has position `1` and so on until the last element has position `n-1` in an array that contains `n` elements in total.

Problem: Picking elements out of arrays

1. Select the third element of all three, x, y, and z.
2. Select the 11^{th} element of x.
3. Using double index notation, select the (0,2) and the (2,0) element of x.

Issues to ponder

- Which index is rows and which index is columns?
- Does NumPy count across first then down or down first then across?

Problem: Selecting Entire Rows

1. Select the 2nd row of x using the colon (:) operator.
2. Select the 2nd element of z and y using the same syntax.

Issues to ponder

- What happens to the output in each case?

Problem: Selecting Entire Columns

Select the 2nd column of x using the colon (:) operator.

Problem: Selecting Specific Rows or Columns

1. Select the 2nd and 3rd columns of x using the colon (:) operator.
2. Select the 2nd and 4th rows of x.
3. Combine these be combined to select columns 2 and 3 and rows 2 and 4.

Problem: Use `ix_` to select arbitrary rows and columns

Use `ix_` to select the 2nd and 4th rows and 1st and 3rd columns of `x`.

Problem: Convert a DataFrame to a NumPy array

Use `.to_numpy` to convert a DataFrame to a NumPy array.

```
# Setup: Create a DataFrame
import pandas as pd
import numpy as np

names = ['a', 'b', 'c', 'd', 'e']
x = np.arange(25).reshape((5,5))
x_df = pd.DataFrame(x, index=names, columns=names)
print(x_df)
```

Problem: Use `np.asarray` to convert to an array

Use `np.asarray` to convert a DataFrame toa NumPy array.

Lesson 10

Numeric Indexing of DataFrames

This lesson covers:

- Accessing specific elements in DataFrames using numeric indices

Accessing elements in a DataFrame is a common task. To begin this lesson, clear the workspace set up some vectors and a 5×5 array. These vectors and matrix will make it easy to determine which elements are selected by a command.
Begin by creating:

- A 5-by-5 DataFrame `x_df` containing `np.arange(25).reshape((5,5))`.
- A 5-element Series `y_s` containing `np.arange(5)`.
- A 5-by-5 DataFrame `x_named` that is `x_df` with columns 'c0', 'c1', ..., 'c4' and rows 'r0', 'r1', ..., 'r4'.
- A 5-element Series `y_named` with index 'r0', 'r1', ..., 'r4'.

Problem: Picking an Element out of a DataFrame

Using double index notation, select the (0,2) and the (2,0) element of `x_named`.

Problem: Selecting Entire Rows

1. Select the 2nd row of `x_named` using the colon (:) operator.
2. Select the 2nd element of `y_named` using the same syntax.

Problem: Selecting Entire Columns

Select the 2nd column of `x_named` using the colon (:) operator.

Problem: Selecting Specific Columns

Select the 2nd and 3rd columns of `x_named` using the colon (:) operator.

Problem: Select Specific Rows

Select the 2nd and 4th rows of `x_named`. Combine these be combined to select columns 2 and 3 and rows 2 and 4.

Problem: Select Specific Rows and Columns

Combine these be combined to select columns 2 and 3 and rows 2 and 4.

Problem: Select arbitrary rows and columns

Select the 2nd and 4th rows and 1st and 3rd columns of `x`.

Note: This is the only important difference with NumPy. Arbitrary row/column selection using `DataFrame.iloc` is simpler but less flexible.

Problem: Mix names and integer selection

Select the columns c1 and c2 and rows 0, 2 and 4.

Problem: Mix names and integer selection

Select the rows r1 and r2 and columns 0, 2 and 4.

Lesson 11

Program Flow

This lesson covers:

- for loops
- Nested loops

Problem: Basic For Loops

Construct a for loop to sum the numbers between 1 and N for any N. A for loop that does nothing can be written:

```
n = 10
for i in range(n):
    pass
```

Problem: Compute a compound return

The compound return on a bond that pays interest annually at rate r is given by $c\,r_t = \prod_{i=1}^{T}(1+r) = (1+r)^T$. Use a for loop compute the total return for £100 invested today for $1, 2, \ldots, 10$ years. Store this variable in a 10 by 1 vector cr.

Problem: Simulate a random walk

(Pseudo) Normal random variables can be simulated using the command `np.random.standard_normal(shape)` where `shape` is a tuple (or a scalar) containing the dimensions of the desired random numbers. Simulate 100 normals in a 100 by 1 vector and name the result `e`. Initialize a vector `p` containing zeros using the function zeros. Add the 1st element of `e` to the first element of `p`. Use a for loop to simulate a process $y_i = y_{i-1} + e_i$. When finished plot the results using

```
%matplotlib inline

import matplotlib.pyplot as plt
plt.plot(y)
```

Problem: Nested Loops

Begin by loading momentum data used in an earlier lesson. Begin by adding 1 to the returns to produce gross returns. The gross return is the total value in the current period of £1 invested in the previous period. A net return subtracts the original investment to produce the net gain or loss. Use two loops to loop both across time and across the 10 portfolios to compute the total compound return.

For example, if only interested in a single series,

```
n = mom_01.shape[0]
cr=np.zeros(n)
gr = 1 + mom_01
cr[0] = 1+mom_01[0]
for t in range(1, n):
    cr[t]=cr[t-1]*gr[t]
```

computes the cumulative return.

When finished, plot the cumulative returns using `plt.plot(cr)`. After finishing this problem, have a look at `np.cumsum?` and `np.cumprod?`.

```
# Setup: Load the momentum data

import pandas as pd
momentum = pd.read_csv('data/momentum.csv', index_col='date', parse_dates=True)
momentum = momentum / 100  # Convert to numeric values from percentages
# Convert to a plain numpy array
momentum = momentum.to_numpy()
```

Exercises

Exercise

1. Simulate a 1000 by 10 matrix consisting of 10 standard random walks using both nested loops and `np.cumsum`.
2. Plot the results.

Question to think about

If you rerun the code in this Exercise, do the results change? Why?

Lesson 12

Logical Operators

This lesson covers:

- Basic logical operators
- Compound operators

Begin by loading the data in momentum.csv.

```
# Setup: Load the momentum data

import pandas as pd

momentum = pd.read_csv('data/momentum.csv', index_col='date', parse_dates=True)

print(momentum.head())

mom_01 = momentum['mom_01']
mom_10 = momentum['mom_10']
mom_05 = momentum['mom_05']
```

Problem: Basic Logical Statements

For portfolio 1 and portfolio 10, count the number of elements that are < 0, ≥ 0, and exactly equal to 0. Next count the number of times that the returns in portfolio 5 are greater, in absolute value, that 2 times the standard deviation of the returns in that portfolio.

Problem: Compound Statements

Count the number of times that the returns in both portfolio 1 and portfolio 10 are negative. Next count the number of times that the returns in portfolios 1 and 10 are both greater, in absolute value, that 2 times their respective standard deviations.

Lesson 13

Logic and Loops

This lesson covers:

- Mixing logic and loops
- `all` and `any`

Begin by loading the data in momentum.csv.

```
# Setup: Load the momentum data

import numpy as np
import pandas as pd

momentum = pd.read_csv('data/momentum.csv', index_col='date', parse_dates=True)

print(momentum.head())

mom_01 = momentum['mom_01']
mom_10 = momentum['mom_10']
mom_05 = momentum['mom_05']
```

Problem: Logical Statements and for Loops

Use a for loop along with an `if` statement to simulate an asymmetric random walk of the form

$$y_i = y_{i-1} + e_i + I_{[e_i<0]}e_i$$

where $I_{[e_i<0]}$ is known as an indicator variable that takes the value 1 if the statement in brackets is true. Plot y. e is a standard normal shock. Use `cumsum` to simulate a symmetric one (`z`), and plot the two using the code in the cell below.

Problem: Simulate the asymmetricc random walk without an `if-then`

Use boolean multiplication to simulate the same random walk without using an `if-then` statement.

```
# Setup: Plot the data
%matplotlib inline
```

Problem: Selecting Elements using Logical Statements

For portfolio 1 and portfolio 10, select the elements that are < 0, ≥ 0 and exactly equal to 0. Next select the elements where both portfolios are less than 0.

Problem: Using `where`

Use `where` to select the index of the elements in portfolio 5 that are negative. Next, use the `where` command in its two output form to determine which elements of the portfolio return matrix are less than -2%.

Problem: Combining flow control

For momentum portfolios 1 and 10, compute the length of the runs in the series. In pseudo code,

- Start at i=1 and define run(1) = 1
- For i in 2,...,T, define run(i) = run(i-1) + 1 if $\text{sgn}(r_i) = \text{sgn}(r_{i-1})$ else 1.

You will need to use `len` and `zeros`.

1. Compute the length longest run in the series and the index of the location of the longest run. Was it positive or negative?
2. How many distinct runs lasted 5 or more days?

Problem: Use any to find large losses

Use any to determine if any of the 10 portfolios experienced a loss greater than -5%.

Use `all` and negation to do the same check as `any`.

Exercises

Exercise: all and any

Use all to determine the number of days where all of the portfolio returns were negative. Use any to compute the number of days with at least 1 negative return and with no negative returns (Hint: use negation (~ or `logical_not`)).

Lesson 14

Importing Data

This lesson covers:

- Importing data
- Converting dates

Problem: Reading in data with Dates

Read in the files `GS10.csv` and `GS10.xls` which have both been downloaded from FRED.

Problem: Converting Dates

1. Load the CSV file without converting the dates in `read_csv`.
2. Convert the date column, remove it from the DataFrame, and set it as the index.

Lesson 15

Saving and Exporting Data

This lesson covers:

- Saving and reloading data

This first block loads the data that was used in the previous lesson.

```
# Setup: Load the data to use later
import pandas as pd

gs10_csv = pd.read_csv('data/GS10.csv', index_col='DATE', parse_dates=True)
gs10_excel = pd.read_excel('data/GS10.xls', skiprows=10,
                           index_col='observation_date')
```

Problem: Export to Excel

Export `gs10_csv` to the Excel file `gs10-exported.xlsx`.

Problem: Export to Excel sheets

Export both `gs10_excel` and `gs10_csv` to the same Excel file.

Problem: Export to CSV

Export `gs10_excel` to CSV.

Problem: Export to HDF

Export both to an HDF file (the closest thing to a "native" format in pandas)

Problem: Export to Pickle

1. Export `gs10_excel` to a pickle file.
2. Combine `gs10_excel` and `gs10_csv` into a dictionary and pickle the dictionary.

Problem: Import from HDF and Pickle.

Import the data saved in steps 3-5 of the previous problem.

Lesson 16

Graphics: Line Plots

This lesson covers:

- Basic plotting
- Subplots
- Histograms
- Scatter Plots

Plotting in notebooks requires using a magic command, which starts with `%`, to initialize the plotting backend.

```
# Setup
%matplotlib inline
```

Begin by loading the data in hf.h5. This data set contains high-frequency price data for IBM and MSFT on a single day stored as two Series. IBM is stored as 'IBM' in the HDF file, and MSFT is stored as 'MSFT.

Problem: Basic Plotting

1. Plot the `ibm` series which contains the price of IBM.
2. Add a title and label the axes.
3. Add markers and remove the line.

Problem: Subplot

Create a 2 by 1 subplot with the price of IBM in the top subplot and the price of MSFT in the bottom subplot.

Problem: Plot with Dates

Use `matplotlib` to directly plot `ibm` against its `index`. This is a repeat of a previous plot but shows how to use the `plot` command directly.

Lesson 17

Graphics: Other Plots

This lesson covers:

- Histograms
- Scatter Plots

Plotting in notebooks requires using a magic command, which starts with `%`, to initialize the plotting backend.

```
# Setup
%matplotlib inline
```

Begin by loading the data in hf.h5. This data set contains high-frequency price data for IBM and MSFT on a single day stored as two Series. IBM is stored as 'IBM' in the HDF file, and MSFT is stored as 'MSFT.

Problem: Histogram

Produce a histogram of MSFT 1-minute returns (Hint: you have to produce the 1-minute Microsoft returns first using `resample` and `pct_change`).

Problem: Scatter Plot

Scatter the 5-minute MSFT returns against the 5-minute IBM returns.

Hint: You will need to create both 5-minute return series, merge them, and then plot using the combined DataFrame.

Problem: Saving plots

Save the previous plot to PNG and PDF.