A very brief Introduction to Python for <u>Data Science</u>
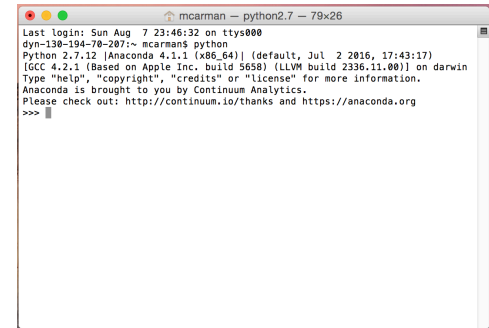
Updated 16/8/2016

What is Python?

- Python is a general purpose programming language
  - interpreted (aka scripting) language -- like Javascript & R
  - designed by computer scientists (not statisticians)
  - open-source
  - very popular

- Alternatives:
  - R, Matlab, SAS, …

Getting started

- Anaconda extends Python with various packages useful for Data Science

- Install Python+Anaconda from:
  - https://www.continuum.io/downloads

- Or use Python online:
  - by opening a new iPython notebook on Jupyter
  - https://try.jupyter.org/
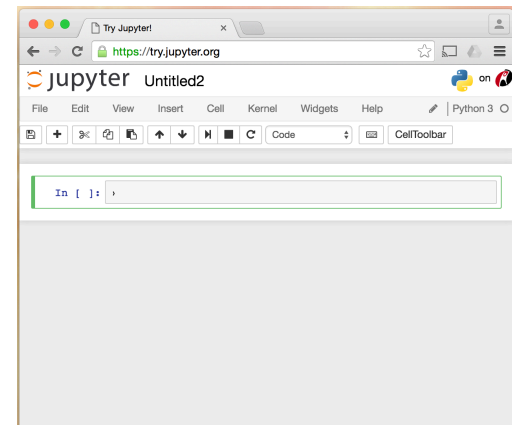
Basic Python syntax

- Compute mathematical expressions:

```
> 3+2
5
```

- Define variables and assign values:

```
> A = 10
> 5*A + 6
56
```

- Define a list:

```
> B = [5,5,3,0]
> B
5 5 3 0
```

- Access part of list:

```
> B[0]
5
> B[1:3]
[5,3]
```

Here > denotes the command prompt

Formula syntax comes from C, so some expressions are interpreted differently from R:
- e.g. to compute 2 cubed type $2**3$ not 2^3

Can use square or round brackets:
```
> B = (5,5,3,0)
```

BEWARE: First index of array is 0 not 1!

Loading libraries

- Load a library with "from … import … as …" keywords

```
> from matplotlib import pyplot as plt
> plt.boxplot(…)
```

plt is the name we are giving to the imported library.
We use it whenever we want to call a function provided by the library

Can also use shorter "." notation rather than "from" to load library:
```
import matplotlib.pyplot as plt
```

Data Tables in Python

- The pandas library provides a table structure called DataFrame

```
> import pandas as pd
> df = pd.DataFrame({
    'StudentID' : [264422,264423,264444,259432],
    'Name' : ['Steven','Alex','Bill','Steven'],
    'Mark' : [93.5,61.2,78.5,81.1]
})
```

Creates a table with 3 columns:
- StudentID, Name and Mark

- Print out the table table:

```
> print(df)
    Mark     Name   StudentID
0   93.5   Steven      264422
1   61.2     Alex      264423
2   78.5     Bill      264444
3   81.1   Steven      259432
```
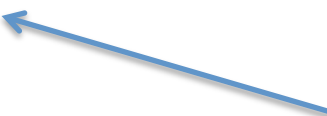
Selecting Columns and values

- Select a column by using its column name:

```
> df['Name']
0     Steven
1       Alex
2       Bill
3     Steven
Name: Name, dtype: object
```

- Select **multiple columns** using an **list** of column names:

```
> df[['Name','Mark']]
     Name    Mark
0   Steven   93.5
1     Alex   61.2
2     Bill   78.5
3   Steven   81.1
```

Note the need for the double bracket, since the argument is now an array

- Select a value using the column name and row index:

```
> df['Name'][1]
'Alex'
```

Selecting Rows

- Select a particular row from the table:
```
> df.loc[2]
Mark              78.5
Name              Bill
StudentID       264444
Name: 2, dtype: object
```

To select multiple rows, replace the single index by an array:
```
> df.loc[[0,2]]
    Mark      Name   StudentID
0   93.5    Steven      264422
2   78.5      Bill      264444
```

- Select **all** rows with a particular value in one of the columns:
```
> df.loc[df['Name'] == 'Steven']
    Mark      Name   StudentID
0   93.5    Steven      264422
3   81.1    Steven      259432
```

Loading & Saving Data

- We can loading data in from a CSV file

```
> import pandas as pd
> df = pd.read_csv('input.csv',sep=',')
```

- And save a resulting data frame to a CSV file

```
> df2 = df.loc[df['Name'] == 'Steven']
> df2.save_csv('output.csv')
```

Aggregation and Groupby

- Sometimes we need compute aggregate values (like totals or averages) for a particular column:

```
> df['Mark'].sum()
314.29999999999995
> df['Mark'].mean()
78.57499999999999
```

- And often we'd like to know these aggregate values for certain values of another column, in which case we use the *groupby* function:

```
> df.groupby('Name')['Mark'].mean()
Name
Alex       61.2
Bill       78.5
Steven     87.3
Name: Mark, dtype: float64
```

Average value for two students named Steven from original table:

	Mark	Name	StudentID
0	93.5	Steven	264422
1	61.2	Alex	264423
2	78.5	Bill	264444
3	81.1	Steven	259432

Aggregation and Groupby (cont.)

- In order to "flatten" the results of the *groupby* function back into a table use the *reset_index* method:

```
> df1 = df.groupby('Name')['Mark'].mean()
> df1 = df1.reset_index()
     Name  Mark
0    Alex  61.2
1    Bill  78.5
2  Steven  87.3
```

- We can aggregate with respect to **multiple attributes at once** by inserting an array of column names into the *groupby* call:

```
> df.groupby(['Name','StudentID'])['Mark'].mean()
Name    StudentID
Alex    264423         61.2
Bill    264444         78.5
Steven  259432         81.1
        264422         93.5
Name: Mark, dtype: float64
```

Mean values for particular **combinations** of Name and StudentID

- Finally, we can return the aggregated values (mean or sum) for multiple columns at once by listing the columns explicitly or not at all:

```
> df.groupby(['Name'])['Mark','StudentID'].mean()
> df.groupby(['Name']).mean()
```

Slides by Mark Carman

Plotting data

- We can use the matplotlib library to plot data in Python

```
import matplotlib.pylab as plt
```
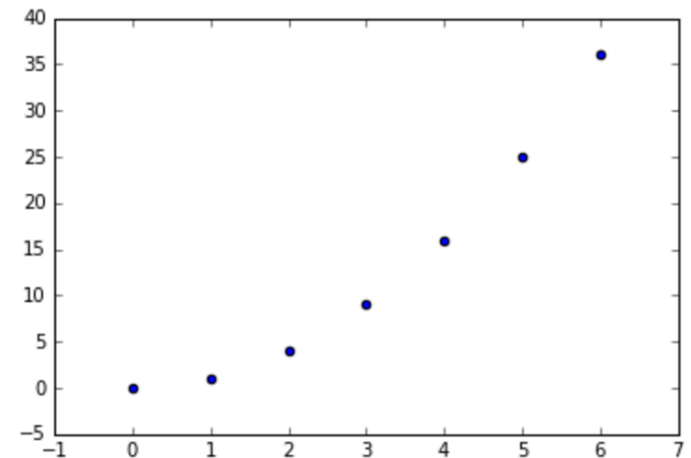
- Define a table with the data to plot:

```
> df = pd.DataFrame({
'X' : [0,1,2,3,4,5,6],
'Y' : [0,1,4,9,16,25,36]})
```

- Create scatter plot

```
> plt.scatter(df['X'], df['Y'])
```

- And show it:

```
plt.show()
```

End of Introduction

- We'll be playing around with Python in this week's Tutorial

- There are MANY excellent Python resources online if you'd like to learn more
  - for example: lynda.com