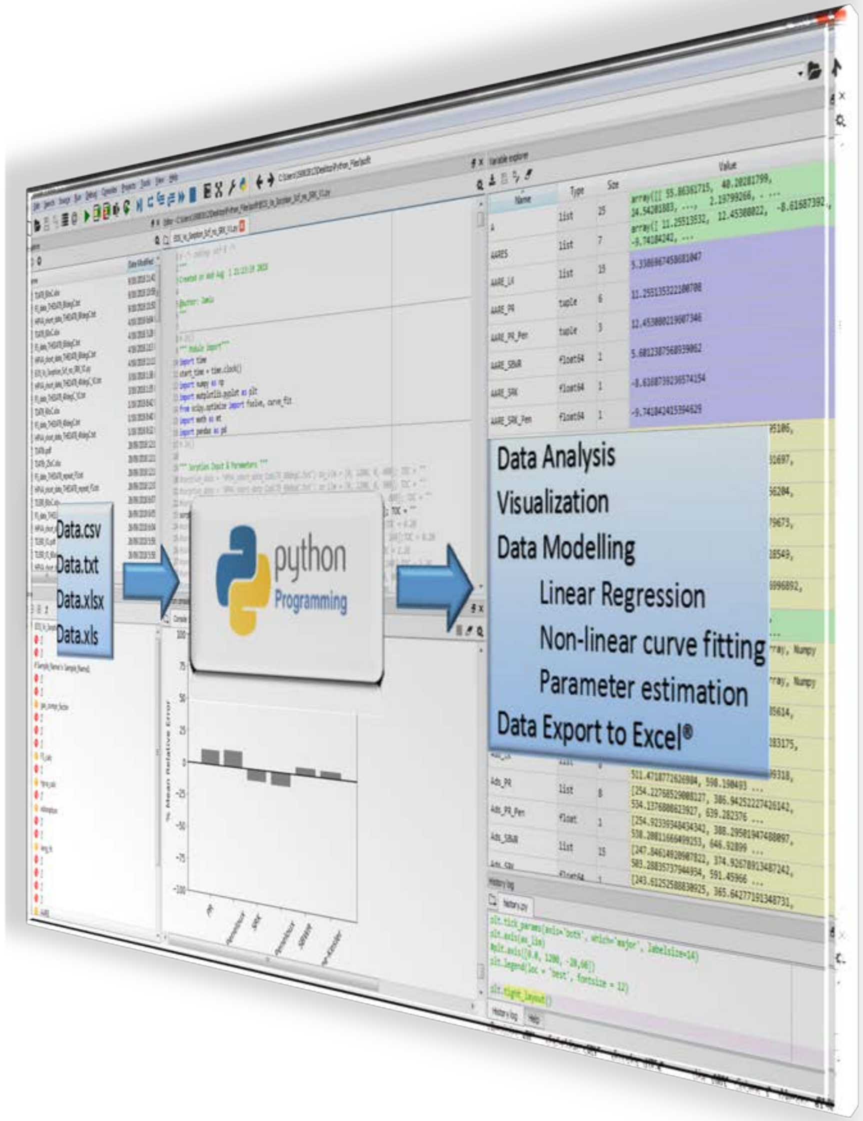
Python Crash Course

Research Data Analysis, Visualisation & Modelling with Python



Delivered

By

Jamiu M. Ekundayo
PhD Candidate
Petroleum Eng. Dept.
Curtin University

Learning Objectives

- At the end of this workshop, participants should be able to:
  - ✓ Read input data from files (*.txt, *.csv, etc.)
  - ✓ Explore, visualize and analyse data
  - ✓ Perform data modelling (LSQ optimization, curve-fitting, parameter estimation, etc.)
  - ✓ Generate inputs for Excel®

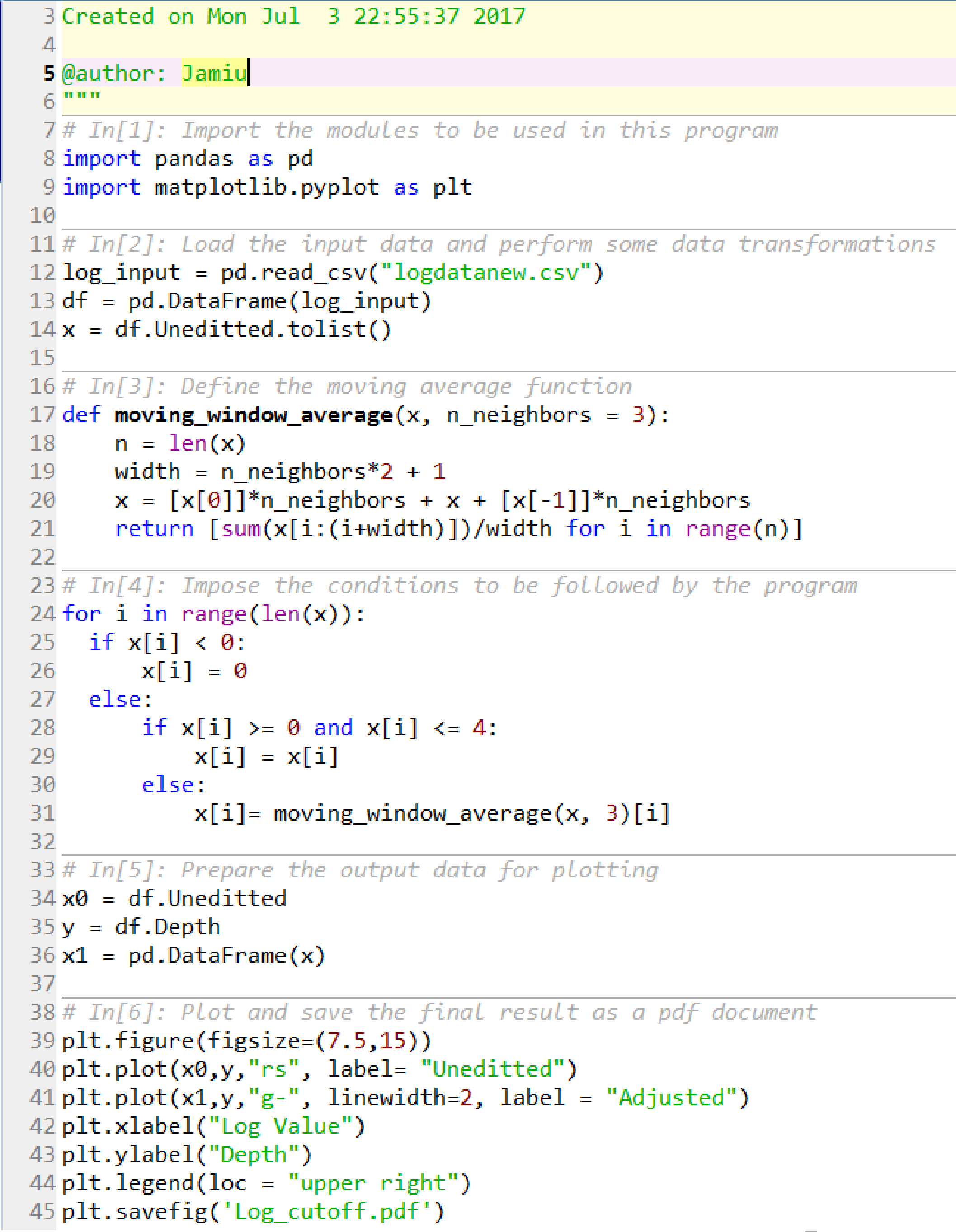
Selected Projects

Project Name	Description	Expected Outcomes	Input Files
1. Text Analysis	Analysis of the Gettysburg Address by Abraham Lincoln in 1863	Demonstrate the application of python basics, user-defined functions, visualisation, etc.	Abraham Lincoln.txt
2. Parameter Estimation	Estimation of the optimal parameters by curve-fitting (non-linear regression) some data to a function	In addition to the outcomes in project 1, this project will enable participants to demonstrate application of selected python modules to solving scientific problems.	project2_data .txt

Course Contents & Proposed Timing

- Session 1 – 10.00am – 11.30am
  - Overview & Python basics
    - ✓ Operators, data types, variables & in-built functions
    - ✓ Working with conditions & loops
    - ✓ Functions (user-defined)
    - ✓ Data I/O (reading from & writing to files)
    - ✓ Working with modules
    - ✓ Data visualization
- Session 2 – 11.30am – 12noon
  - Project 1 – Text analysis
- Session 3 – 1.00pm – 3.00pm
  - Data Exploration & Modelling
    - ✓ Working with structured data
    - ✓ Linear & non-linear regressions
    - ✓ Curve-fitting and parameter estimation
    - ✓ Data Export to Excel®
- Session 4 – 3.00pm – 4.00pm
  - Project 2 – parameter estimation

Sample Python Program

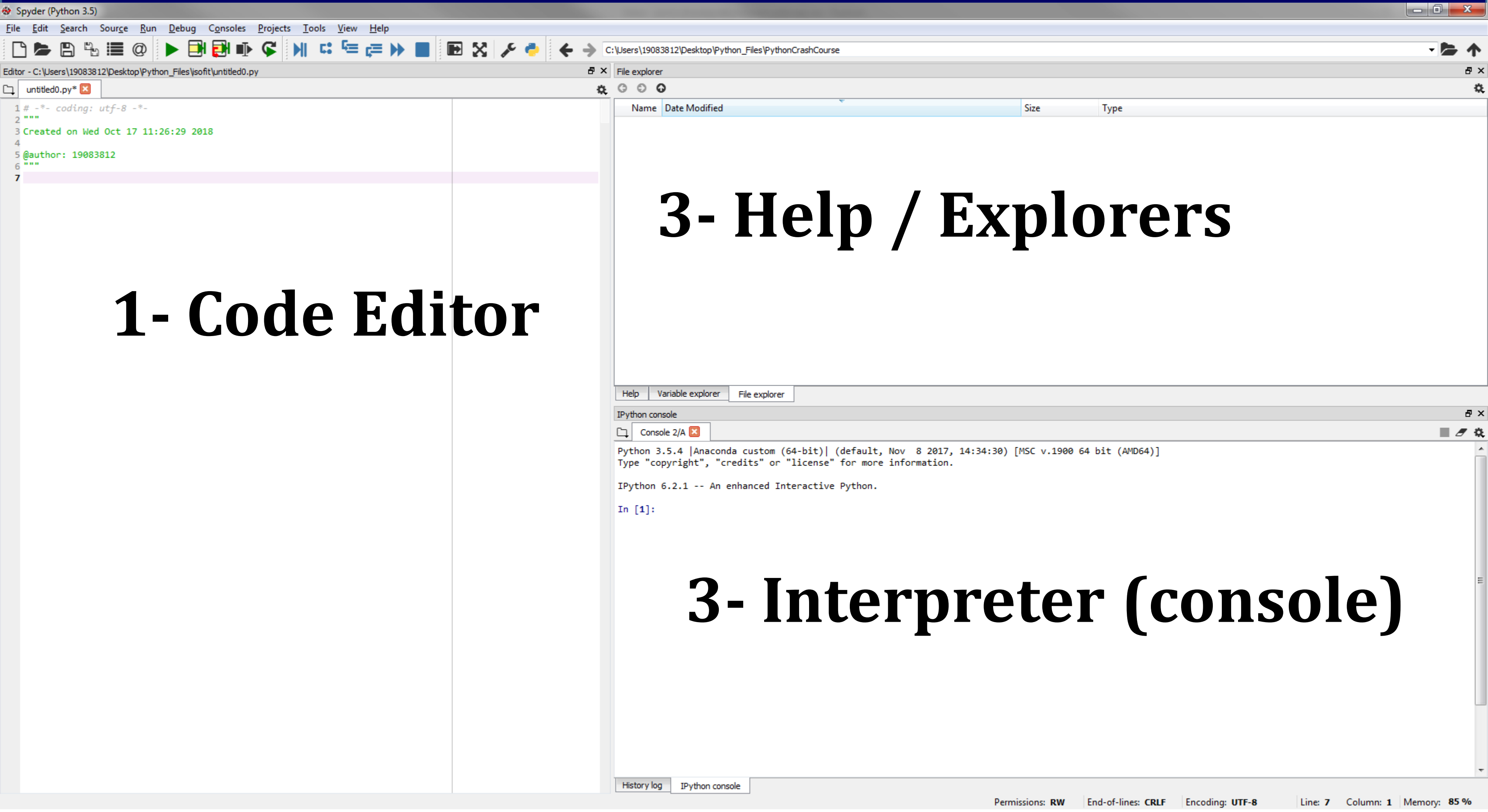


```
Created on Mon Jul  3 22:55:37 2017

@author: Jamiu
"""
# In[1]: Import the modules to be used in this program
import pandas as pd
import matplotlib.pyplot as plt

# In[2]: Load the input data and perform some data transformations
log_input = pd.read_csv("logdatanew.csv")
df = pd.DataFrame(log_input)
x = df.Uneditted.tolist()

# In[3]: Define the moving average function
def moving_window_average(x, n_neighbors = 3):
    n = len(x)
    width = n_neighbors*2 + 1
    x = [x[0]]*n_neighbors + x + [x[-1]]*n_neighbors
    return [sum(x[i:(i+width)])/width for i in range(n)]

# In[4]: Impose the conditions to be followed by the program
for i in range(len(x)):
  if x[i] < 0:
      x[i] = 0
  else:
      if x[i] >= 0 and x[i] <= 4:
          x[i] = x[i]
      else:
          x[i]= moving_window_average(x, 3)[i]

# In[5]: Prepare the output data for plotting
x0 = df.Uneditted
y = df.Depth
x1 = pd.DataFrame(x)

# In[6]: Plot and save the final result as a pdf document
plt.figure(figsize=(7.5,15))
plt.plot(x0,y,"rs", label= "Uneditted")
plt.plot(x1,y,"g-", linewidth=2, label = "Adjusted")
plt.xlabel("Log Value")
plt.ylabel("Depth")
plt.legend(loc = "upper right")
plt.savefig('Log_cutoff.pdf')
```

Session 1

Overview
&
Python Basics

Spyder – Default Layout


Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\19083812\Desktop\Python_Files\PythonCrashCourse
Editor - C:\Users\19083812\Desktop\Python_Files\isofit\untitled0.py
untitled0.py*
File explorer
Name Date Modified Size Type
Help Variable explorer File explorer
IPython console
Console 2/A
History log IPython console
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 7 Column: 1 Memory: 85 %

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 17 11:26:29 2018

@author: 19083812
"""
```

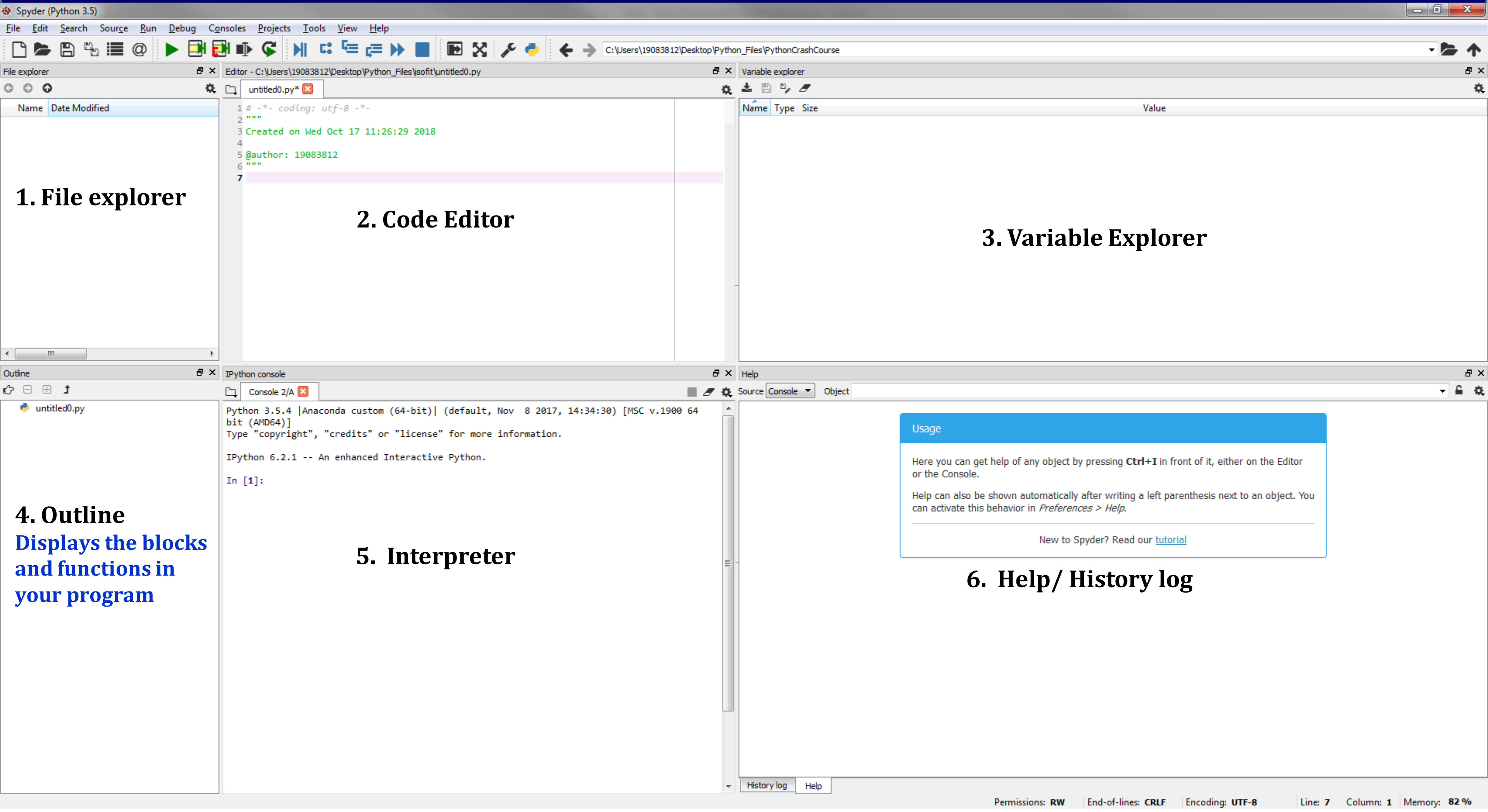
```
Python 3.5.4 |Anaconda custom (64-bit)| (default, Nov  8 2017, 14:34:30) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]:
```

1- Code Editor

3- Help / Explorers

3- Interpreter (console)

Spyder – Matlab Layout



Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\19083812\Desktop\Python_Files\PythonCrashCourse
File explorer
Name Date Modified
Editor - C:\Users\19083812\Desktop\Python_Files\isofit\untitled0.py
untitled0.py*
Variable explorer
Name Type Size Value
Outline
untitled0.py
IPython console
Console 2/A
Help
Source Console Object
History log Help
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 7 Column: 1 Memory: 82 %

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 17 11:26:29 2018

@author: 19083812
"""
```

```
Python 3.5.4 |Anaconda custom (64-bit)| (default, Nov  8 2017, 14:34:30) [MSC v.1900 64
bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]:
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

1. File explorer

2. Code Editor

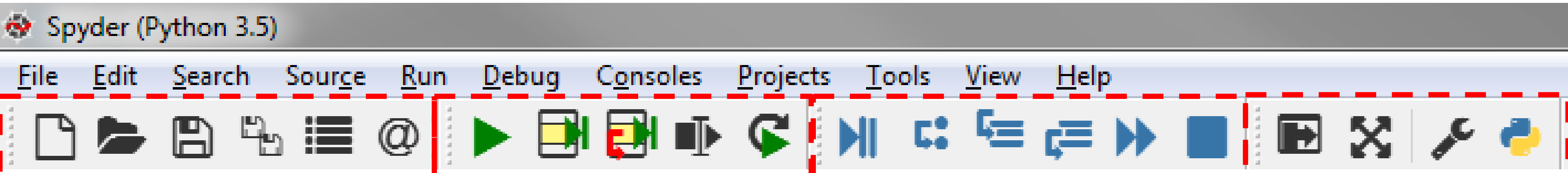
3. Variable Explorer

4. Outline
Displays the blocks and functions in your program

5. Interpreter

6. Help/ History log

Menu Items


Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help

File Management Pane

- Create new
- Switch between files
- Save files
- Inspect files

Execution Pane

- Run entire program
- Run current code block
- Run current code cell & proceed to next
- Run selected codes
- Re-run last file

Debugging Pane

- Not used in this course

Preference Pane

- Maximize/restore windows
- Full-screen
- Settings

Python Operators

Operator	Description	Example
+	Addition	1 + 2 = 3
-	Subtraction	2-1 = 1
*	Multiplication	1*2 = 2
/	Division	1/2 = 0.5
**	Power	3**2 = 9
//	Floor/Integer division	1//2 = 0
%	Modulus/remainder	1%2 = 1
_	Underscore* (Returns the last object called by python)	
|	Boolean operator (OR) (Returns 'TRUE' if 1 of 2 or more expressions is true)	(1 < 2) | (3 < 2) = True
&	Boolean operator (AND) (Returns 'TRUE' if all of 2 or more expressions are true)	(1 < 2) & (3 < 2) = False
!	Boolean operator (NOT) (Returns the negation of sets)	1 != 2

* Only available in the interactive mode

NOTE:

- Python follows the mathematical order of precedence – i.e. order of evaluating an expression

Python Operators

Just for fun! Type & run this expression in your IPython Console:

```
5*2+3**2-1/2%3//4
```

Can you figure out how Python arrived at 19.0?

Try!

```
5*2+3**2-1/(2%3)//4
```

Is the result 19.0?

The message here is:

Using parenthesis (brackets) can save you from unforeseen computation errors

Python Data-Types, in-built functions and Variables

Data Types in python: Important to know how python stores an object

Type	Example	Function equivalent
String	"5.0", "I am a string"	str() e.g. str('25')
Integer	5	int() e.g. int(5.75)
Floats	5.0	float() e.g. float(3.67)
Complex	1 + 2j	complex() e.g. complex(-2, 5)

Variables and their Naming Rules

- A variable is string to which an expression is assigned e.g. my_var = 1 + 2
- A variable can:
  - ✓ Only contain one word (no space between letters)
  - ✓ Only use letters, numbers and underscore
  - ✓ Not begin with a number

Python Data-Types, in-built functions and Variables

Common String Operations: Some interesting things you can do with strings
Let’s say will have a string called my_string = ‘What is in a Name?’

Operation	Description	Example
len(string)	Returns the number of characters in a string	len(my_string)
string.split(sep = “ ”)	Returns a list of each item in the string separated by space (or any other specified separator)	my_string.split( )
string.replace(old, new)	Returns a new string with the specified item replaced	my_string.replace(‘is’, ‘is not’)
+	Concatenates a specified number of strings	my_string.replace('?', " ") + 'other than letters?'
string[start:stop:step] This is called **slicing**	Returns a string of characters in the positions start to stop in specified steps	my_string[5:10] my_string[5:10:2]
string[::-1]	Used to reverse a string	my_string[::-1]

You can learn more interactively on this website:
https://www.learnpython.org/en/Basic_String_Operations

Python Data-Types, in-built functions and Variables

Formatting Strings: Particularly useful when floats are being converted to strings for reporting

Format	Description	Example
%s format(input, ‘s’)	Used for strings (texts)	"The first five characters are '%s'" % my_string[:5]
%d format(input, ‘d’)	Used to format integers as strings	'My string is %d long.' % len(my_string) 'My string is '+ format(len(my_string), 'd') + ' long.'
%f format(input, ‘f’)	Used to format floats as strings	'pi = %f' % (22/7) 'pi = ' + format(22/7, 'f')
%.<integer>f format(input, ‘.<integer>f’)	Used to format a float as a string with a specified number of decimal places	'pi = %.4f' % (22/7) 'pi = ' + format(22/7, '.4f')

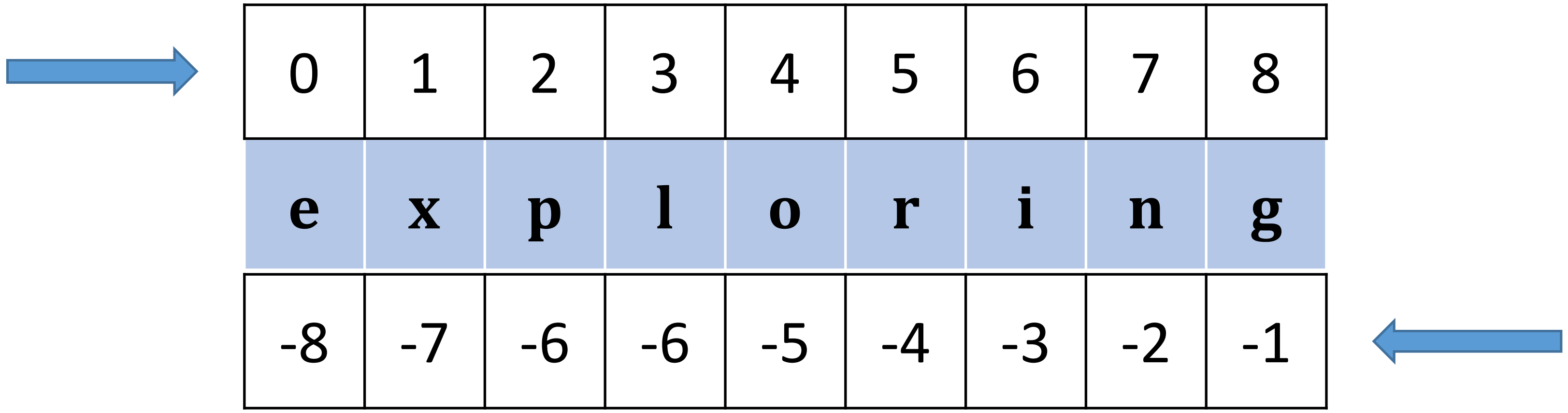
Try this!

```
'A=%.3f, B=%.3f, C=%.3f, D=%.3f' % tuple(numpy.random.randn(4))
```

Python Data-Types, in-built functions and Variables

<u>**Indexing/Slicing**</u>

- Indexing in python starts from 0 and is usually left-to-right although right-to-left indexing is also supported



0	1	2	3	4	5	6	7	8
e	**x**	**p**	**l**	**o**	**r**	**i**	**n**	**g**
-8	-7	-6	-6	-5	-4	-3	-2	-1

- Try these:

  1. My_string = “exploring”
  2. My_string[8]
  3. My_string[-1]

Python Data-Types, in-built functions and Variables

- **Lists, Tuples & Sets:** Sequences of objects of different types
- **Examples**:

  My_list = [1, 2, 'a', 'b', 'c']
  My_tuple = tuple(My_list)
  My_set = set(My_list)

- **Mutability & 'hashability'**? Try the following in your ipython console:

My_list.append('5')	My_list[3]
My_tuple.add('5')	My_tuple[3]
My_set.add('5')	My_set[3]

- **Lists are mutable & hashable, tuples are immutable but hashable while sets are mutable (unless frozen) but not hashable**

- **Try these**:
  - My_froset = frozenset(My_list)
  - My_froset.add('5')

Python Data-Types, in-built functions and Variables

- **Range (range(start, stop, step))**: Immutable sequence of integers.
- **Examples**:
  1. range(5)
  2. range(0, len(my_list), 2)

- **Python Operations/methods with Lists, Tuples and Sets: To find out what you can do with any of these items, simply type help(item) in your console**

Item	Common Operations/Methods
List	List.append( ), List.reverse( ), List.remove( ), List.clear( ), List.remove( ), etc.
Tuple	Tuple.count( ), Tuple.index( )
Set	Set.add( ), Set.remove( ), Set.update( ), Set.copy( ), etc.

- **Examples**: mylist_new = list(range(0, 10, 2))
  1. mylist_new.append(6)
  2. tuple(mylist_new).count(6)

Python Data-Types, in-built functions and Variables

- **Dictionary:** Unordered sequence of mutable objects

- A dictionary has two main attributes namely keys and values. A dictionary is actually formed by mapping the elements of values to the corresponding elements of keys.

- For example:
    - Name = ['Lukman', 'Partha', 'Jimmy', 'Yujie', 'Aja']; Age = [50, 25, 32, 23, 24]
    - Dictionary_1= dict(zip(Name, Age))

- **Some dictionary methods includes**
    - Dictionary_1.key( )
    - Dictionary_1.values( )
    - Dictionary_1.update({"Jie": 27})
    - Dictionary_1["Xiu"] = 31
    - Dictionary_1.pop["Xiu"]
    - Dictionary_1["Lukman"] +=15

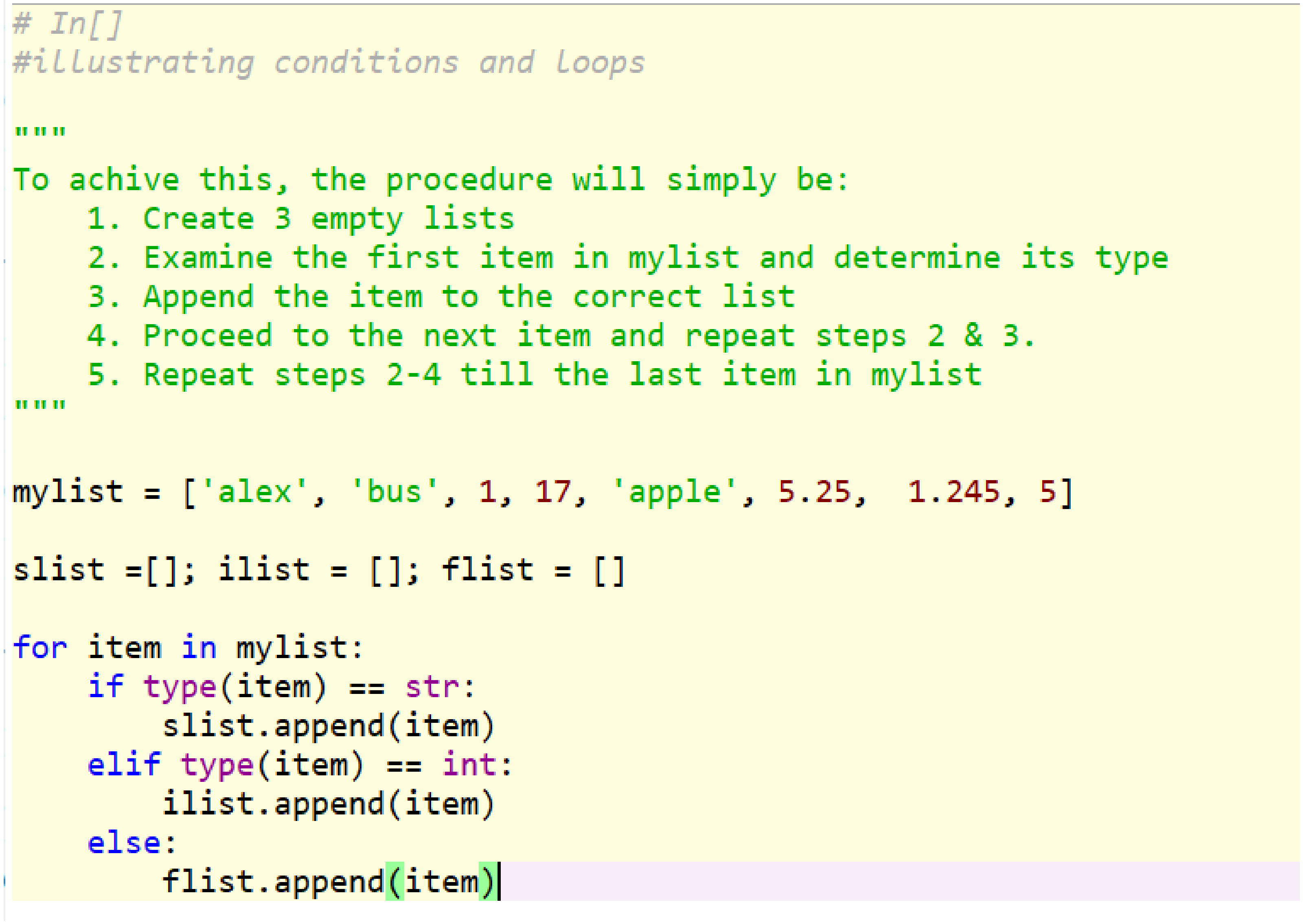
Working with conditions & loops

- "if-elif-else" keywords are used for conditional expressions in python.

```
if  condition1:
    expression1
elif  condition2:
    expression2
:
:
else:
    alternative expression
```

- **Loops** are used for repetitive or iterative operations.
- Such operations are performed using "for … in…" or "while" keywords in python
- **For example**:

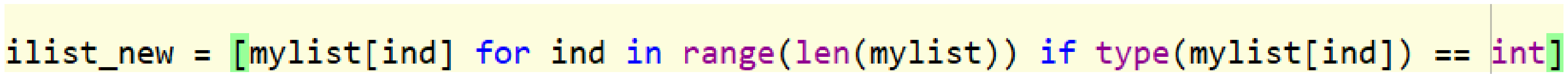
Working with conditions & loops – List Comprehension



```
# In[]
#illustrating conditions and loops

"""
To achive this, the procedure will simply be:
    1. Create 3 empty lists
    2. Examine the first item in mylist and determine its type
    3. Append the item to the correct list
    4. Proceed to the next item and repeat steps 2 & 3.
    5. Repeat steps 2-4 till the last item in mylist
"""

mylist = ['alex', 'bus', 1, 17, 'apple', 5.25,  1.245, 5]

slist =[]; ilist = []; flist = []

for item in mylist:
    if type(item) == str:
        slist.append(item)
    elif type(item) == int:
        ilist.append(item)
    else:
        flist.append(item)
```

List Comprehension

- Assuming the last example actually requires only a list of integers. We simply can do this:



```
ilist_new = [mylist[ind] for ind in range(len(mylist)) if type(mylist[ind]) == int]
```

- This is known as list comprehension. It helps you achieve efficient codes and shorter run times

- A list comprehension is made up of the following components:
  1. Output expression (mylist[ind])
  2. Variable (ind)
  3. Input sequence (range(len(mylist)))
  4. Optional predicate (if type(mylist[ind]) == int)

Functions (user-defined)

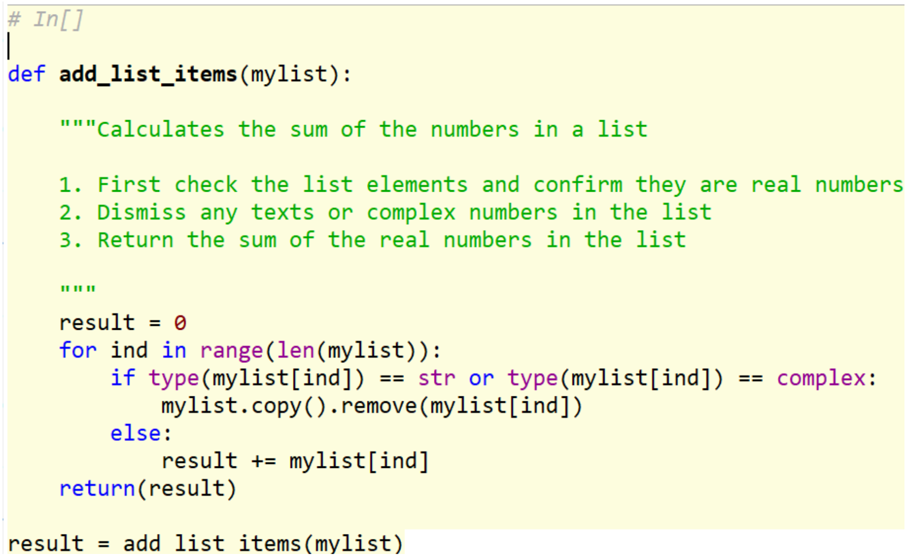
- A function is a block of codes that can be called to perform some task(s)
- The general syntax for user-defined functions in python is:

```
def function_name(parameters):
    """ comments """
    expression
    return(object)
```

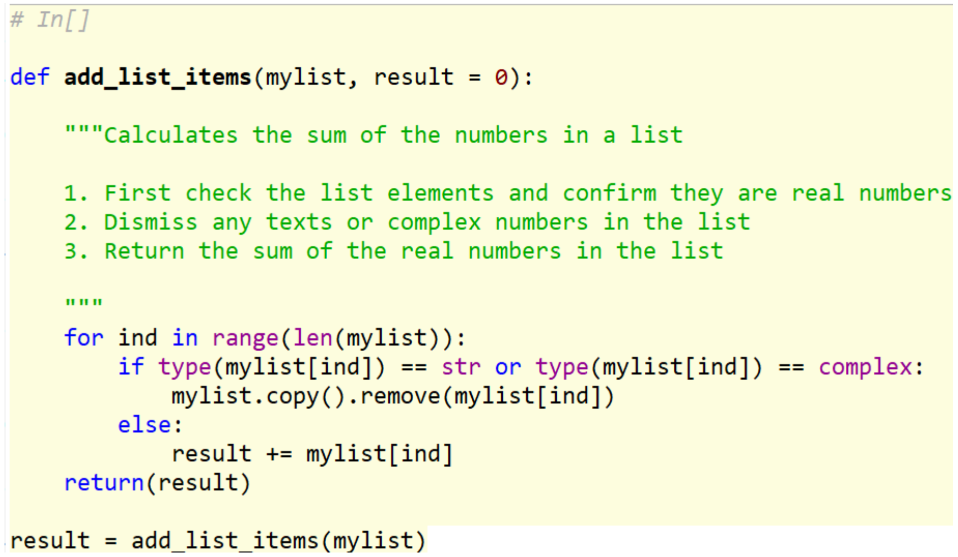
- A function can be called by typing its name and specifying the parameters (if any)

For example: Var1 = function_name(parameters)

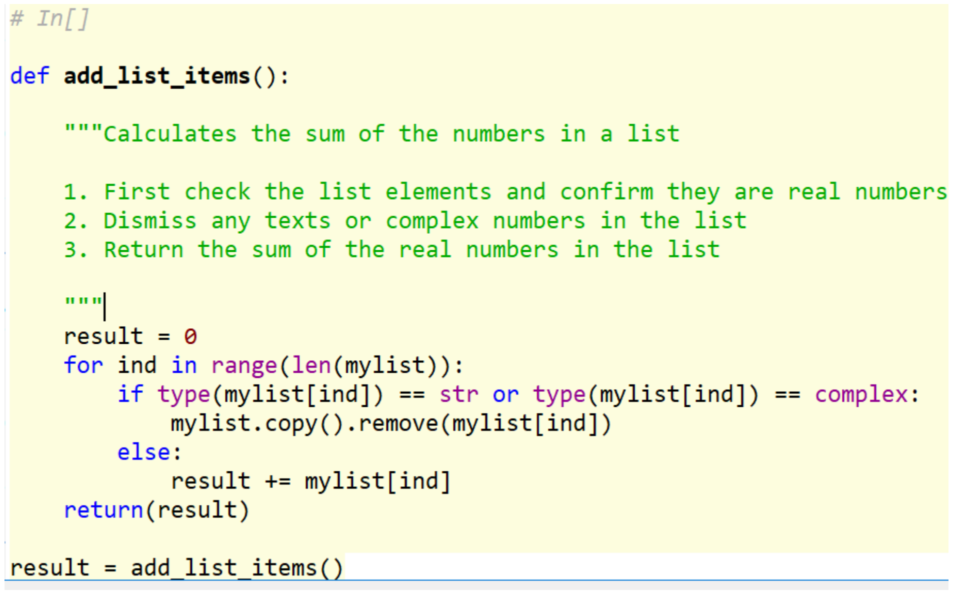
Functions (user-defined)



```python
# In[]

def add_list_items(mylist):

    """Calculates the sum of the numbers in a list

    1. First check the list elements and confirm they are real numbers
    2. Dismiss any texts or complex numbers in the list
    3. Return the sum of the real numbers in the list

    """
    result = 0
    for ind in range(len(mylist)):
        if type(mylist[ind]) == str or type(mylist[ind]) == complex:
            mylist.copy().remove(mylist[ind])
        else:
            result += mylist[ind]
    return(result)

result = add_list_items(mylist)
```



```python
# In[]

def add_list_items(mylist, result = 0):

    """Calculates the sum of the numbers in a list

    1. First check the list elements and confirm they are real numbers
    2. Dismiss any texts or complex numbers in the list
    3. Return the sum of the real numbers in the list

    """
    for ind in range(len(mylist)):
        if type(mylist[ind]) == str or type(mylist[ind]) == complex:
            mylist.copy().remove(mylist[ind])
        else:
            result += mylist[ind]
    return(result)

result = add_list_items(mylist)
```



```python
# In[]

def add_list_items():

    """Calculates the sum of the numbers in a list

    1. First check the list elements and confirm they are real numbers
    2. Dismiss any texts or complex numbers in the list
    3. Return the sum of the real numbers in the list

    """
    result = 0
    for ind in range(len(mylist)):
        if type(mylist[ind]) == str or type(mylist[ind]) == complex:
            mylist.copy().remove(mylist[ind])
        else:
            result += mylist[ind]
    return(result)

result = add_list_items()
```

Data Input/Output (reading from & writing to files)

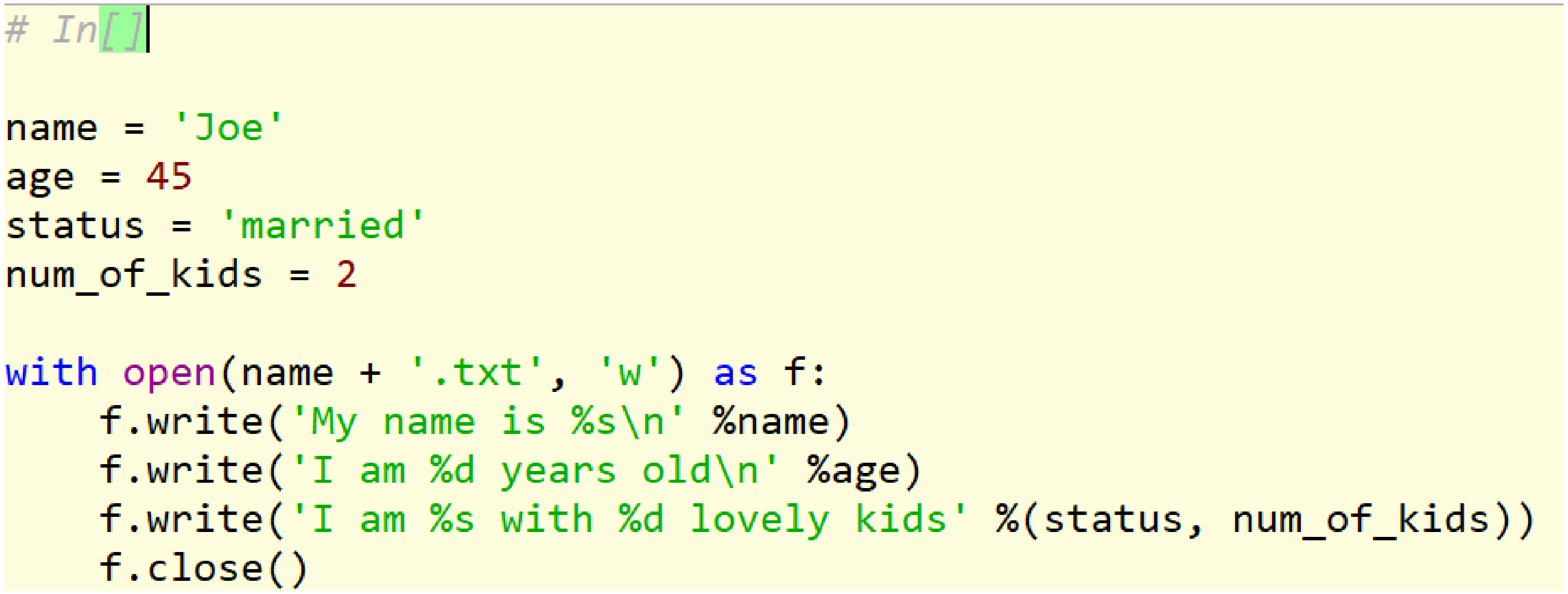
- User input can be obtained using the input( ) function
  e.g. myname = input( )
- To write data to the screen, the function print( ) is used.
  e.g. print('My name is %s' % myname)
- To open file, use the function open(filename, 'r'/'w')
- Alternatively, use with open(filename, 'r'/'w') as name:
- To read data from an open file, use*:
  1. name.read( ) to read the entire content of the file as a string
  2. name.readline( ) to read a single line (usually the first line)
  3. name.readlines( ) to read the file line by line and returns a list of strings
- To write data to a file, the name.write("specify what to write here") is used

*** Care should be taken in using both read & readline(s) together on the same file**

Data Input/Output (reading from & writing to files)

Try the following:

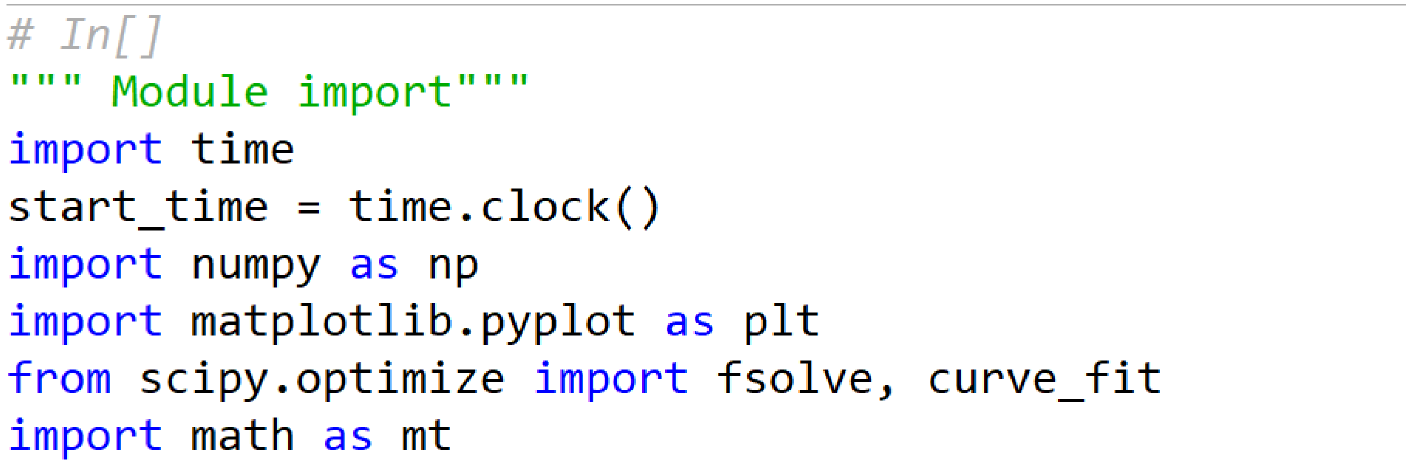
```
file = open('Mary_Lamb.txt', 'r')

file_content = file.read( )

line1 = file.readline( )

lines = file.readlines( )
```



```
# In[ ]

name = 'Joe'
age = 45
status = 'married'
num_of_kids = 2

with open(name + '.txt', 'w') as f:
    f.write('My name is %s\n' %name)
    f.write('I am %d years old\n' %age)
    f.write('I am %s with %d lovely kids' %(status, num_of_kids))
    f.close()
```

Working with modules

- A module is a file containing python definitions and statements that can be deployed to achieve desired results
- Modules are files with extension .py
- A module must be imported prior to use. To import, use the following syntax:
  1. import module_name => This is known as absolute import
  2. from module_name import ABC, XYZ => This is known as relative import



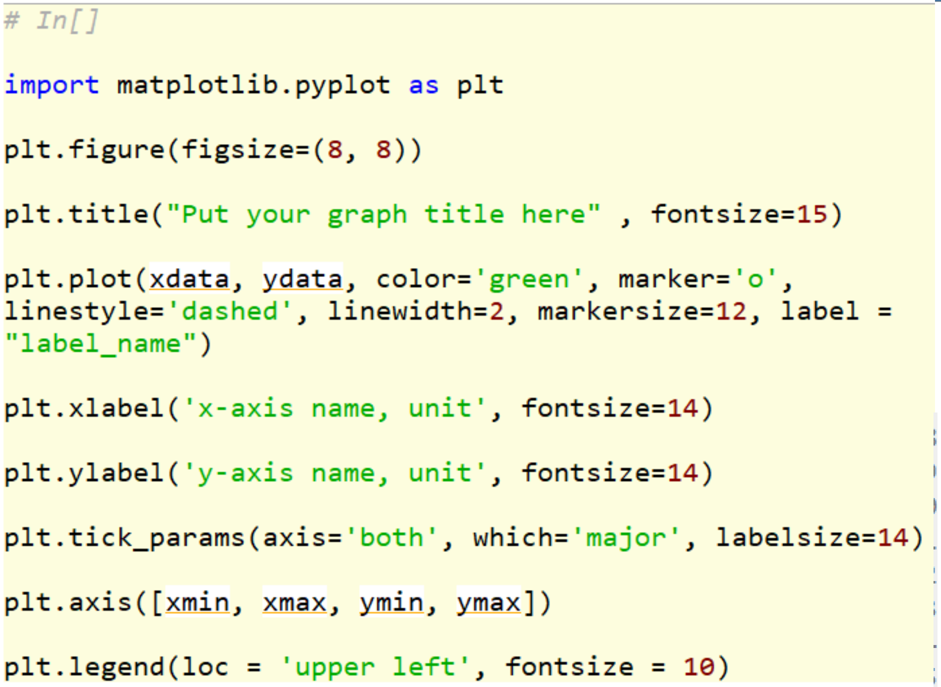
```
# In[]
""" Module import"""
import time
start_time = time.clock()
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve, curve_fit
import math as mt
```

Plotting

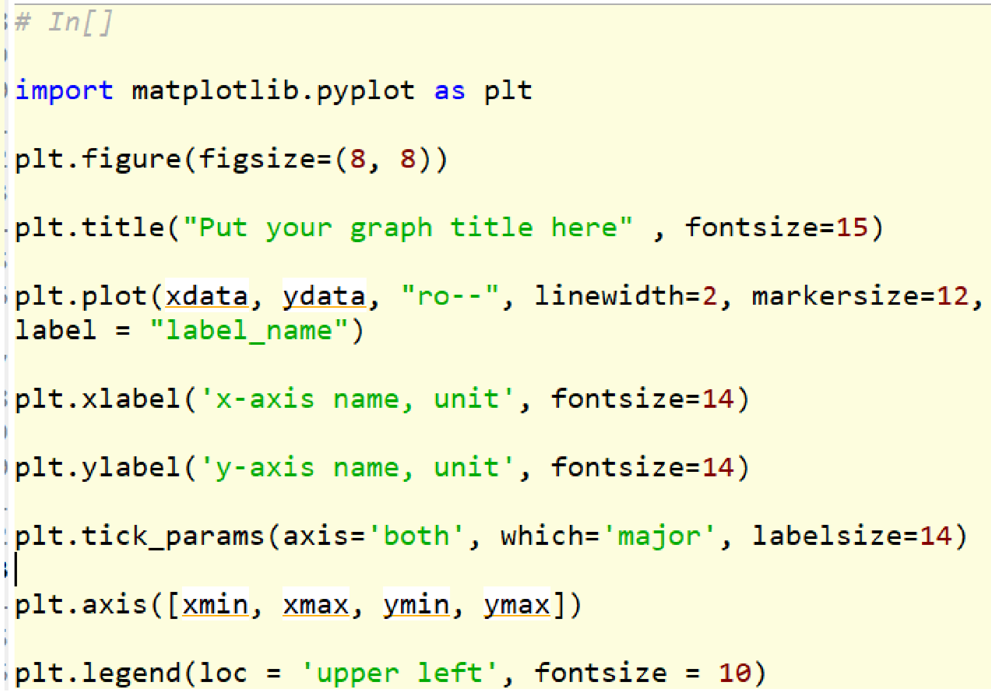
- Most commonly used plotting module in python is matplotlib.pyplot
- To use, simply import matplotlib.pyplot as plt
- There are several plot options available in matplotlib.pyplot
- Examples are:
  - Line plots => plt.plot( )
  - Scatters => plt.scatter( )
  - Histograms => plt.bar( )

- A complete of different plot options are available on https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html

Plotting



```
# In[]

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))

plt.title("Put your graph title here" , fontsize=15)

plt.plot(xdata, ydata, color='green', marker='o',
linestyle='dashed', linewidth=2, markersize=12, label =
"label_name")

plt.xlabel('x-axis name, unit', fontsize=14)

plt.ylabel('y-axis name, unit', fontsize=14)

plt.tick_params(axis='both', which='major', labelsize=14)

plt.axis([xmin, xmax, ymin, ymax])

plt.legend(loc = 'upper left', fontsize = 10)
```



```
# In[]

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))

plt.title("Put your graph title here" , fontsize=15)

plt.plot(xdata, ydata, "ro--", linewidth=2, markersize=12,
label = "label_name")

plt.xlabel('x-axis name, unit', fontsize=14)

plt.ylabel('y-axis name, unit', fontsize=14)

plt.tick_params(axis='both', which='major', labelsize=14)

plt.axis([xmin, xmax, ymin, ymax])

plt.legend(loc = 'upper left', fontsize = 10)
```

Plotting

Abbreviation	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Options	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Marker options	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

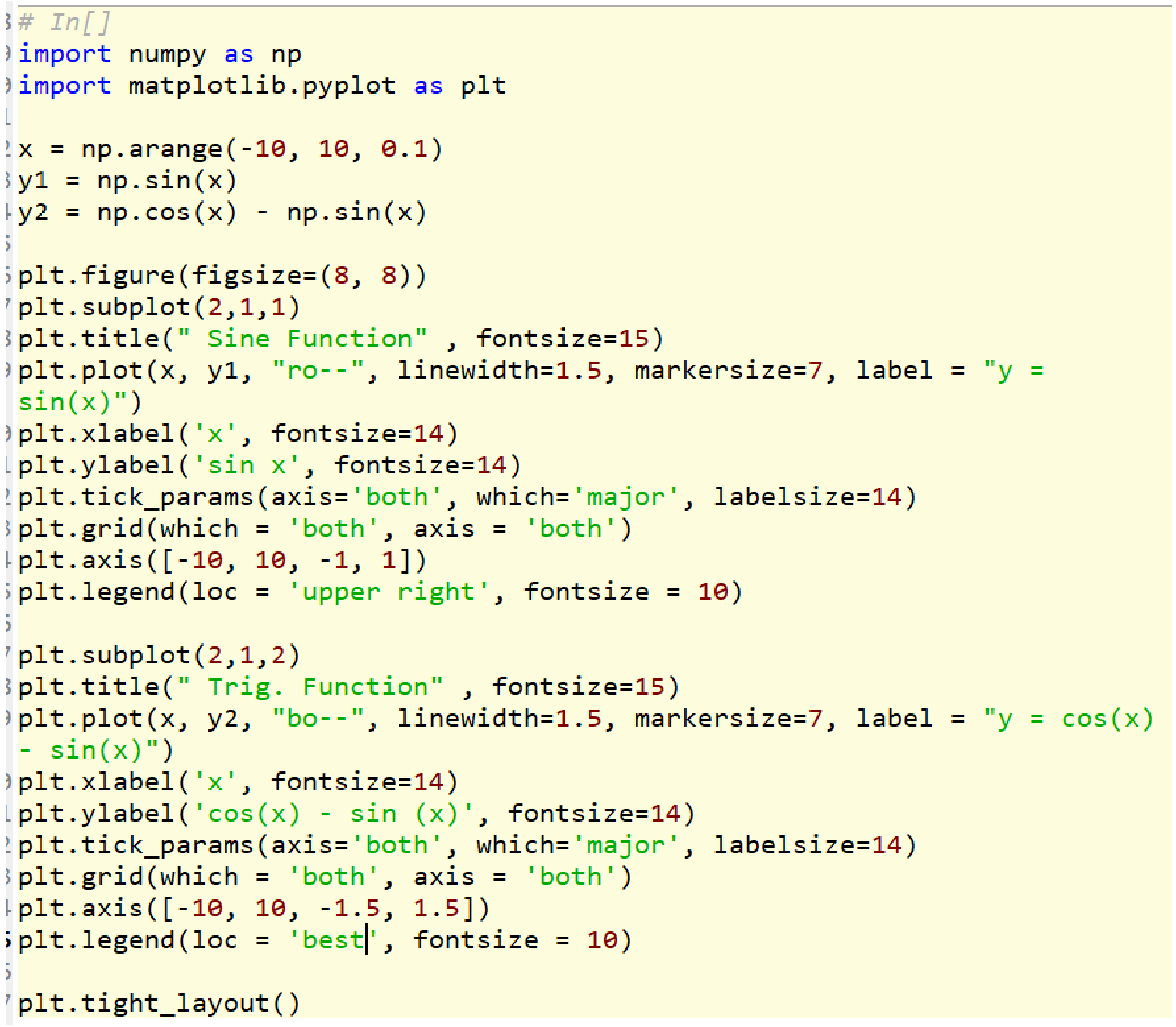
Plotting

```
# In[]
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-10, 10, 0.1)
y1 = np.sin(x)
y2 = np.cos(x) - np.sin(x)

plt.figure(figsize=(8, 8))
plt.subplot(2,1,1)
plt.title(" Sine Function" , fontsize=15)
plt.plot(x, y1, "ro--", linewidth=1.5, markersize=7, label =
"y = sin(x)")
plt.xlabel('x', fontsize=14)
plt.ylabel('sin x', fontsize=14)
plt.tick_params(axis='both', which='major', labelsize=14)
plt.grid(which = 'both', axis = 'both')
plt.axis([-10, 10, -1, 1])
plt.legend(loc = 'upper right', fontsize = 10)
```

Plotting

```
plt.subplot(2,1,2)
plt.title(" Trig. Function" , fontsize=15)
plt.plot(x, y2, "bo--", linewidth=1.5, markersize=7, label =
"y = cos(x) - sin(x)")
plt.xlabel('x', fontsize=14)
plt.ylabel('cos(x) - sin (x)', fontsize=14)
plt.tick_params(axis='both', which='major', labelsize=14)
plt.grid(which = 'both', axis = 'both')
plt.axis([-10, 10, -1.5, 1.5])
plt.legend(loc = 'best', fontsize = 10)

plt.tight_layout()
```

Plotting



```
# In[]
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-10, 10, 0.1)
y1 = np.sin(x)
y2 = np.cos(x) - np.sin(x)

plt.figure(figsize=(8, 8))
plt.subplot(2,1,1)
plt.title(" Sine Function" , fontsize=15)
plt.plot(x, y1, "ro--", linewidth=1.5, markersize=7, label = "y = sin(x)")
plt.xlabel('x', fontsize=14)
plt.ylabel('sin x', fontsize=14)
plt.tick_params(axis='both', which='major', labelsize=14)
plt.grid(which = 'both', axis = 'both')
plt.axis([-10, 10, -1, 1])
plt.legend(loc = 'upper right', fontsize = 10)

plt.subplot(2,1,2)
plt.title(" Trig. Function" , fontsize=15)
plt.plot(x, y2, "bo--", linewidth=1.5, markersize=7, label = "y = cos(x) - sin(x)")
plt.xlabel('x', fontsize=14)
plt.ylabel('cos(x) - sin (x)', fontsize=14)
plt.tick_params(axis='both', which='major', labelsize=14)
plt.grid(which = 'both', axis = 'both')
plt.axis([-10, 10, -1.5, 1.5])
plt.legend(loc = 'best', fontsize = 10)

plt.tight_layout()
```

Session 2

Project 1

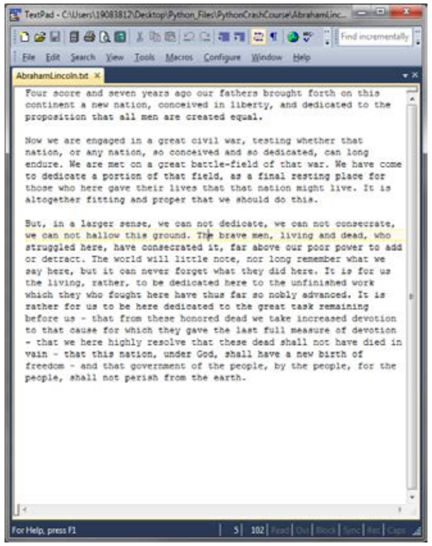
Python Data-Types, in-built functions and Variables

Aim: To consolidate the understanding of python basics and demonstrate how to:

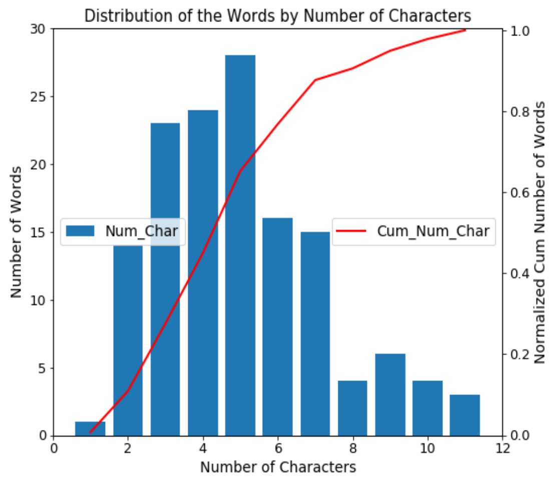
1. Apply python in-built functions
2. Define own functions (user-defined functions)
3. Visualize data and format plots

Description: Analysis of a text input – frequency and length of each word in a given text

Input file: AbrahamLincoln.txt


TextPad - C:\Users\19083812\Desktop\Python_Files\PythonCrashCourse\AbrahamLinc...
Find incrementally
File Edit Search View Tools Macros Configure Window Help
AbrahamLincoln.txt
For Help, press F1

```
Four score and seven years ago our fathers brought forth on this
continent a new nation, conceived in liberty, and dedicated to the
proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that
nation, or any nation, so conceived and so dedicated, can long
endure. We are met on a great battle-field of that war. We have come
to dedicate a portion of that field, as a final resting place for
those who here gave their lives that that nation might live. It is
altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate, we can not consecrate,
we can not hallow this ground. The brave men, living and dead, who
struggled here, have consecrated it, far above our poor power to add
or detract. The world will little note, nor long remember what we
say here, but it can never forget what they did here. It is for us
the living, rather, to be dedicated here to the unfinished work
which they who fought here have thus far so nobly advanced. It is
rather for us to be here dedicated to the great task remaining
before us - that from these honored dead we take increased devotion
to that cause for which they gave the last full measure of devotion
- that we here highly resolve that these dead shall not have died in
vain - that this nation, under God, shall have a new birth of
freedom - and that government of the people, by the people, for the
people, shall not perish from the earth.
```



Curtin University

Session 3

Data Exploration & Modelling

Data Exploration and Modelling (NumPy, Pandas, SciPy)

NumPy

- **NumPy** provides n-dimensional arrays (of floats) for numerical computing
- Matrices can be easily built and manipulated with NumPy
- Like other modules, NumPy must be imported prior to first usage (**import numpy as np**)
- To build a numpy array (following the import as np), simply type

  my_array = np.array([element1, element2, element3, … ,last_element]) or

  my_array = np.array(my_num_list)

  my_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) => 3 x 3 array

- You can examine the number of elements and shape of an array using array_name.size and array_name.shape respectively
- You can reshape an array to form a new array as follows:

  my_new_array = my_array.reshape(shape)

  (Note that shape is a tuple with 2 elements (nrows, ncols))

Data Exploration and Modelling

Some useful NumPy (array) methods

Function	Description
array(object)	Create an array.
asarray(a)	Convert the input to an array.
copy(a)	Return an array copy of the given object.
empty(shape)	Return a new array of given shape and type, without initializing entries.
empty_like(a)	Return a new array with the same shape and type as a given array.
eye(n)	Return a N-D array with ones on the diagonal and zeros elsewhere.
identity(n)	Return the identity array.
ones(shape)	Return a new array of given shape and type, filled with ones.
ones_like(a)	Return an array of ones with the same shape and type as a given array.
zeros(shape)	Return a new array of given shape and type, filled with zeros.
zeros_like(a)	Return an array of zeros with the same shape and type as a given array.
full(shape, fill_value)	Return a new array of given shape and type, filled with fill_value.
full_like(a, fill_value)	Return a full array with the same shape and type as a given array.
arraytolist(a)	Converts an array to a list object
arange(start, stop, interval)	Returns an array of evenly spaced values within a given interval
linspace(start, stop, n)	Returns an array of n numbers from start to stop
logspace(start, stop, n)	Return an array of n numbers spaced evenly on a log scale.

(Adapted from: https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.array-creation.html)

Data Exploration and Modelling

- **NumPy Matrices:** While most numerical operations involve matrices can be done with arrays (for speed), np.arrays are not matrices although they can be converted to matrices (and vice versa)

- NumPy matrices are formed using the **np.matrix( )** function as follows

```
my_matrix = np.matrix(my_array)
```

Selected Matrix Operations

- Consider two matrices:

  1. A = np.matrix(([1, 0, -1], [2, 1, 0], [-1, 2, 1]))
  2. B = np.matrix(([2, -1, 1], [1, 0, 0], [2, 0, 1]))

- We simply can compute

  1. A + B
  2. A – B
  3. A.transpose (or A.T)
  4. A * B
  5. A.dot(B)

- Compare you answers is 4 & 5, are they the same?

Data Exploration and Modelling

- Now, let's consider:

  1. C = np.array([[1, 0, -1], [2, 1, 0], [-1, 2, 1]])
  2. D = np.array(([2, -1, 1], [1, 0, 0], [2, 0, 1]))

- Then try:

  1. C * D
  2. C.dot(D)

C * D is element by element multiplication compared C.dot(D) that is matrix multiplication

- This is known as array **broadcasting**

- Most operations with arrays are performed element by element. If you need matrix operations, it is safer to work with matrices and not arrays.

Data Exploration and Modelling

Array Slicing

- Similar to list and string slicing/indexing, numpy arrays can also be sliced
- General syntax for slicing an array is

  array_name[row_start:row_stop, col_start:col_stop]

- Remember python indexing starts from "zero"
- To illustrate, consider an array: A1 = np.array(range(15)).reshape((3,5))
- Try the following and examine the results (wrt array A1):

1. A1[:]
2. A1[0:2]
3. A1[:, 0:2]
4. A1[:, 2]
5. A1[1:2, 1:2]
6. A1[1:2:, 1:2]
7. A1[2,:]
8. A1[:, -1]

Data Exploration and Modelling

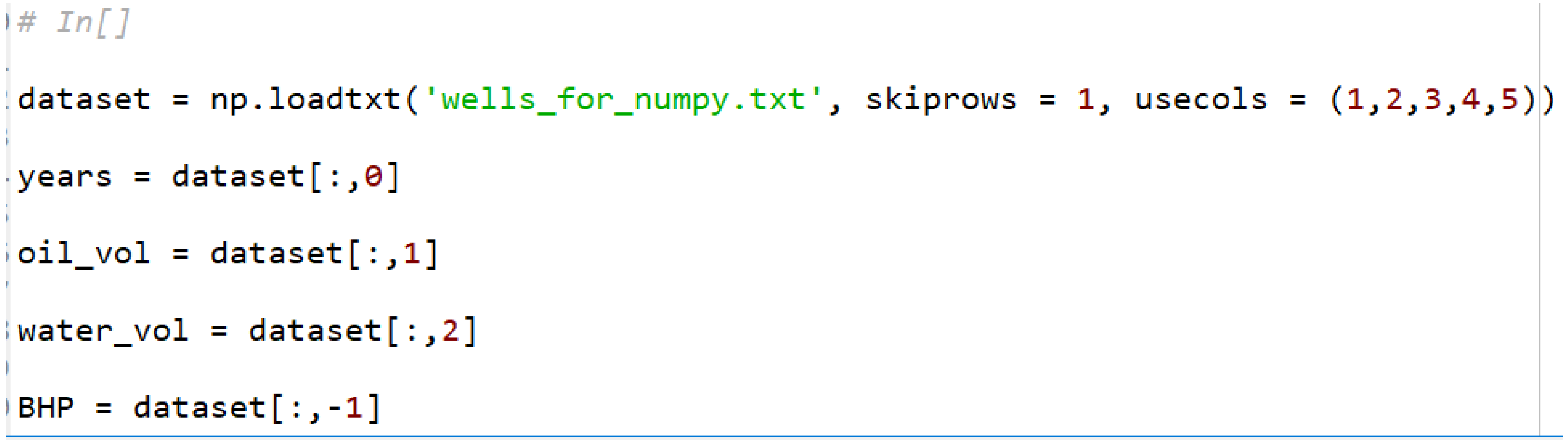
Reading structured data from a file using NumPy

- Structured data (floats) can be read into python as numpy arrays using the np.loadtxt( ) function as follows:

  data_in = np.loadtxt(filename, delimiter = None, skiprows = 0, usecols = None)

- File usually should be any file format readable with a text editor

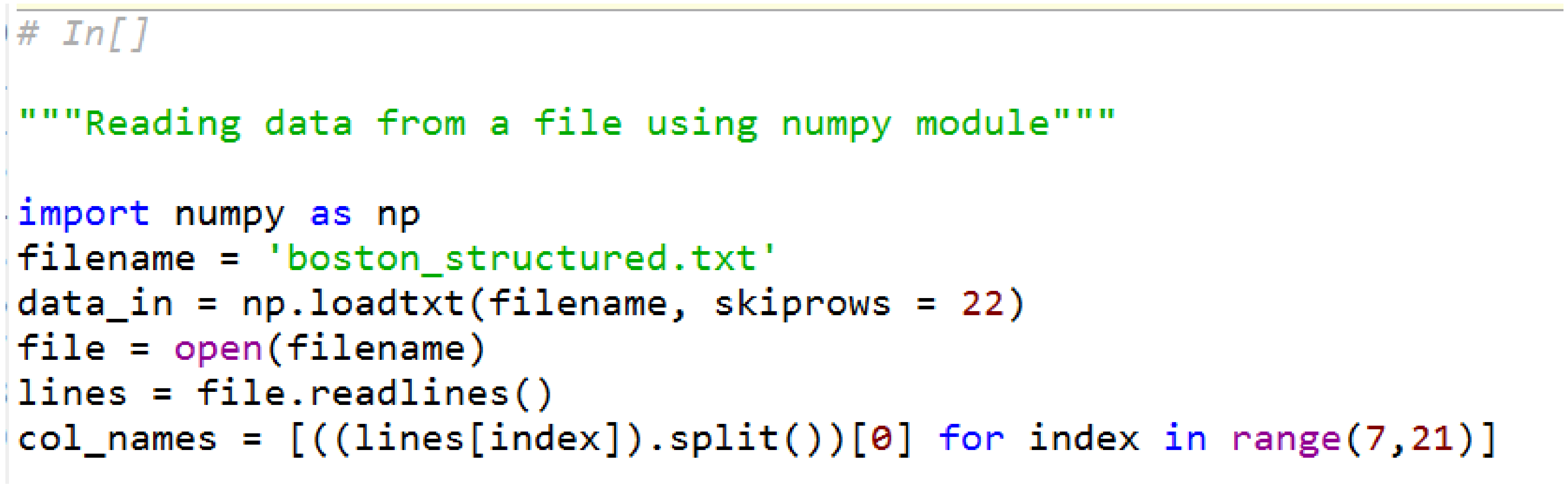
- For example:



```
# In[]

dataset = np.loadtxt('wells_for_numpy.txt', skiprows = 1, usecols = (1,2,3,4,5))

years = dataset[:,0]

oil_vol = dataset[:,1]

water_vol = dataset[:,2]

BHP = dataset[:,-1]
```

Data Exploration and Modelling

- Open the dataset named "boston_structured.txt" with any text editor and examine the contents
- Now, let's open the file with numpy.loadtxt( ) and use open( ) & readlines( ) functions to get the column names – this will be particularly useful in log data analysis

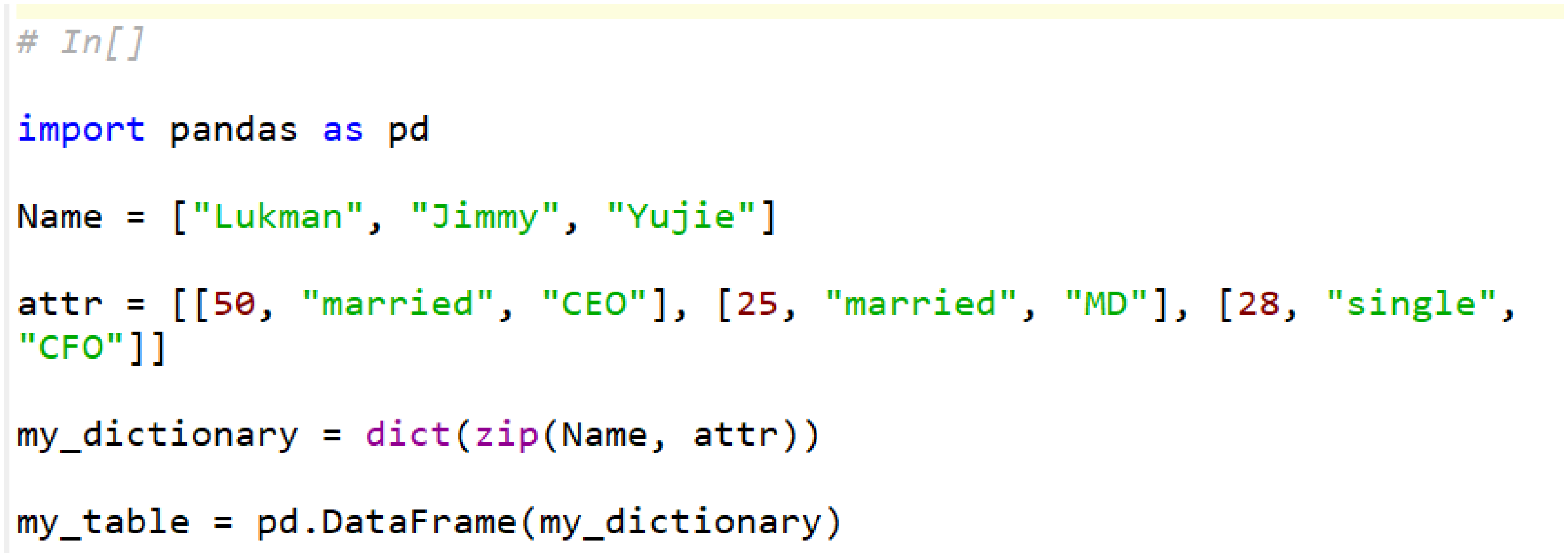


```
# In[]

"""Reading data from a file using numpy module"""

import numpy as np
filename = 'boston_structured.txt'
data_in = np.loadtxt(filename, skiprows = 22)
file = open(filename)
lines = file.readlines()
col_names = [((lines[index]).split())[0] for index in range(7,21)]
```

Data Exploration and Modelling (NumPy, Pandas, SciPy)

<u>Pandas</u>

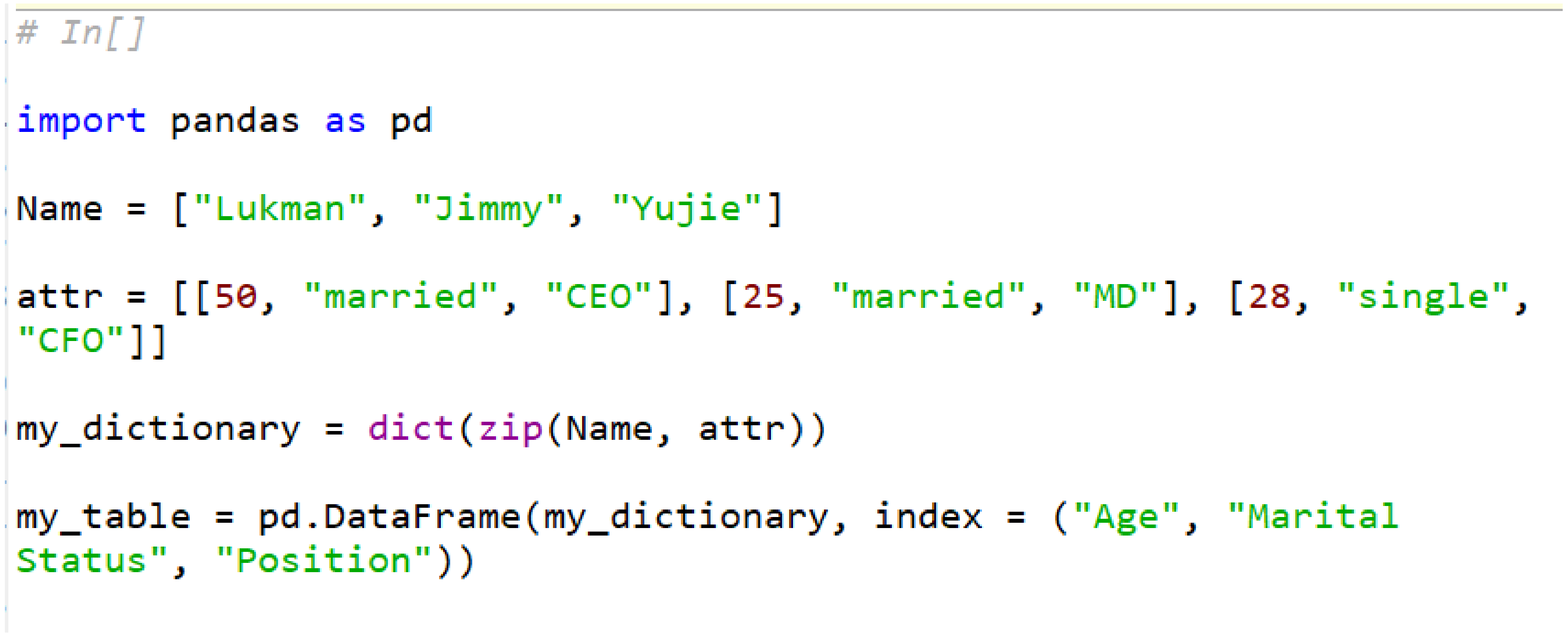
- **Pandas** is useful for manipulating numerical tables and time series data
- It can be used to create tables using the DataFrame function. This is especially useful for generating data for use in Excel
- Like other modules, pandas must be imported prior to first use
- Example:



```
# In[]

import pandas as pd

Name = ["Lukman", "Jimmy", "Yujie"]

attr = [[50, "married", "CEO"], [25, "married", "MD"], [28, "single",
"CFO"]]

my_dictionary = dict(zip(Name, attr))

my_table = pd.DataFrame(my_dictionary)
```

Data Exploration and Modelling (NumPy, Pandas, SciPy)

- The index can be changed to any entry. For example:

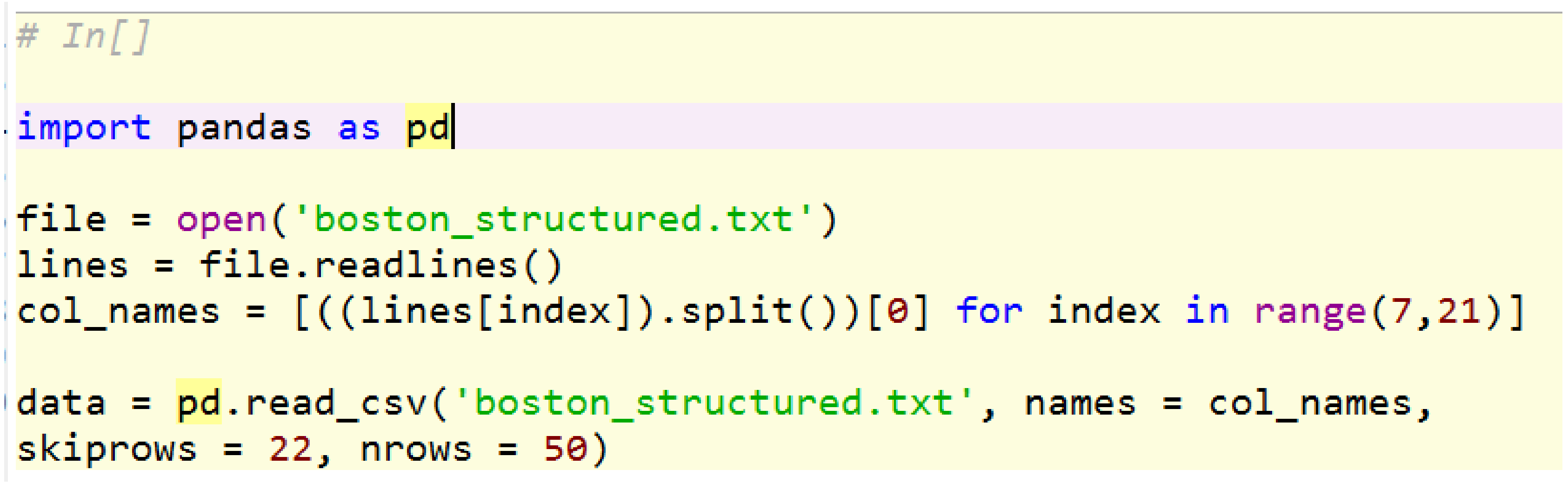


```
# In[]

import pandas as pd

Name = ["Lukman", "Jimmy", "Yujie"]

attr = [[50, "married", "CEO"], [25, "married", "MD"], [28, "single",
"CFO"]]

my_dictionary = dict(zip(Name, attr))

my_table = pd.DataFrame(my_dictionary, index = ("Age", "Marital
Status", "Position"))
```

Data Exploration and Modelling (NumPy, Pandas, SciPy)

- To read data from files using pandas: csv files, tables, excel files, json, html, SAS, SQL, etc.

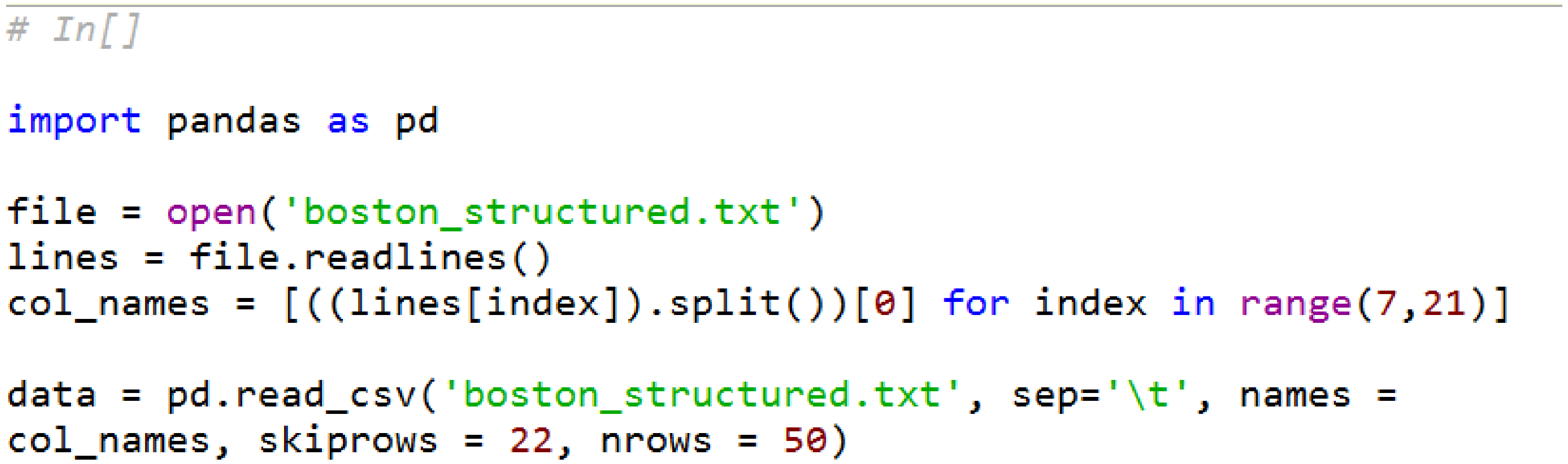
pandas.**read_csv**(*filepath_or_buffer, sep=', ', header='infer', names=None, index_col=None, usecols=None, skipinitialspace=False, skiprows=None, nrows=None, na_values=None, na_filter=True, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, thousands=None, skipfooter=0, delim_whitespace=False*)



```
# In[]

import pandas as pd

file = open('boston_structured.txt')
lines = file.readlines()
col_names = [((lines[index]).split())[0] for index in range(7,21)]

data = pd.read_csv('boston_structured.txt', names = col_names,
skiprows = 22, nrows = 50)
```

- Now, try type data in your console. What can you see?

Data Exploration and Modelling (NumPy, Pandas, SciPy)



```
# In[]

import pandas as pd

file = open('boston_structured.txt')
lines = file.readlines()
col_names = [((lines[index]).split())[0] for index in range(7,21)]

data = pd.read_csv('boston_structured.txt', sep='\t', names =
col_names, skiprows = 22, nrows = 50)
```

- Again, try type data in your console. Any difference?

Data Exploration and Modelling (NumPy, Pandas, SciPy)

```
# In[]

import pandas as pd

file = open('boston_structured.txt')
lines = file.readlines()
col_names = [((lines[index]).split())[0] for index in range(7,21)]

data = pd.read_csv('boston_structured.txt', sep='\t', names =
col_names, skiprows = 22, nrows = 50)
```

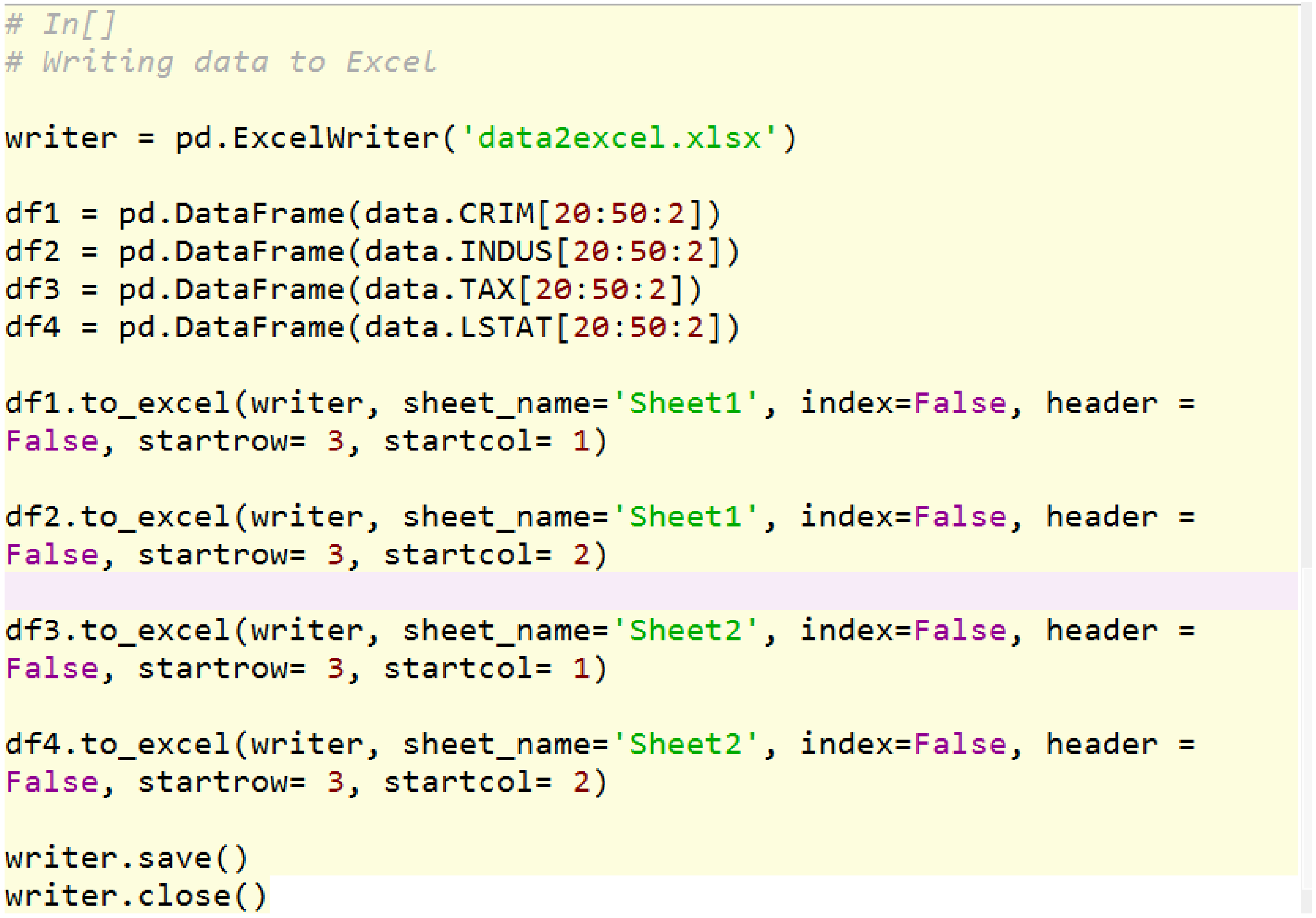
- Again, try type data in your console. Any difference?

Data Exploration and Modelling (NumPy, Pandas, SciPy)

- Now, let's explore the data

- Try:
    1. data.head( )
    2. data.tail( )
    3. data.CRIM
    4. par = np.min(data.CRIM)*data.B
    5. new_array = data.NOX[:-1:2]

- The pandas.ExcelWriter( ) function is used to convert DataFrames to Excel files



```
# In[]
# Writing data to Excel

writer = pd.ExcelWriter('data2excel.xlsx')

df1 = pd.DataFrame(data.CRIM[20:50:2])
df2 = pd.DataFrame(data.INDUS[20:50:2])
df3 = pd.DataFrame(data.TAX[20:50:2])
df4 = pd.DataFrame(data.LSTAT[20:50:2])

df1.to_excel(writer, sheet_name='Sheet1', index=False, header =
False, startrow= 3, startcol= 1)

df2.to_excel(writer, sheet_name='Sheet1', index=False, header =
False, startrow= 3, startcol= 2)

df3.to_excel(writer, sheet_name='Sheet2', index=False, header =
False, startrow= 3, startcol= 1)

df4.to_excel(writer, sheet_name='Sheet2', index=False, header =
False, startrow= 3, startcol= 2)

writer.save()
writer.close()
```

Data Exploration and Modelling

SciPy

- **SciPy** is the scientific computing module in python. It is built from different python modules (most significant NumPy)

- SciPy can be used to solve linear and non-linear problems, perform optimization, statistical analysis, curve-fitting & parameter estimation, etc.

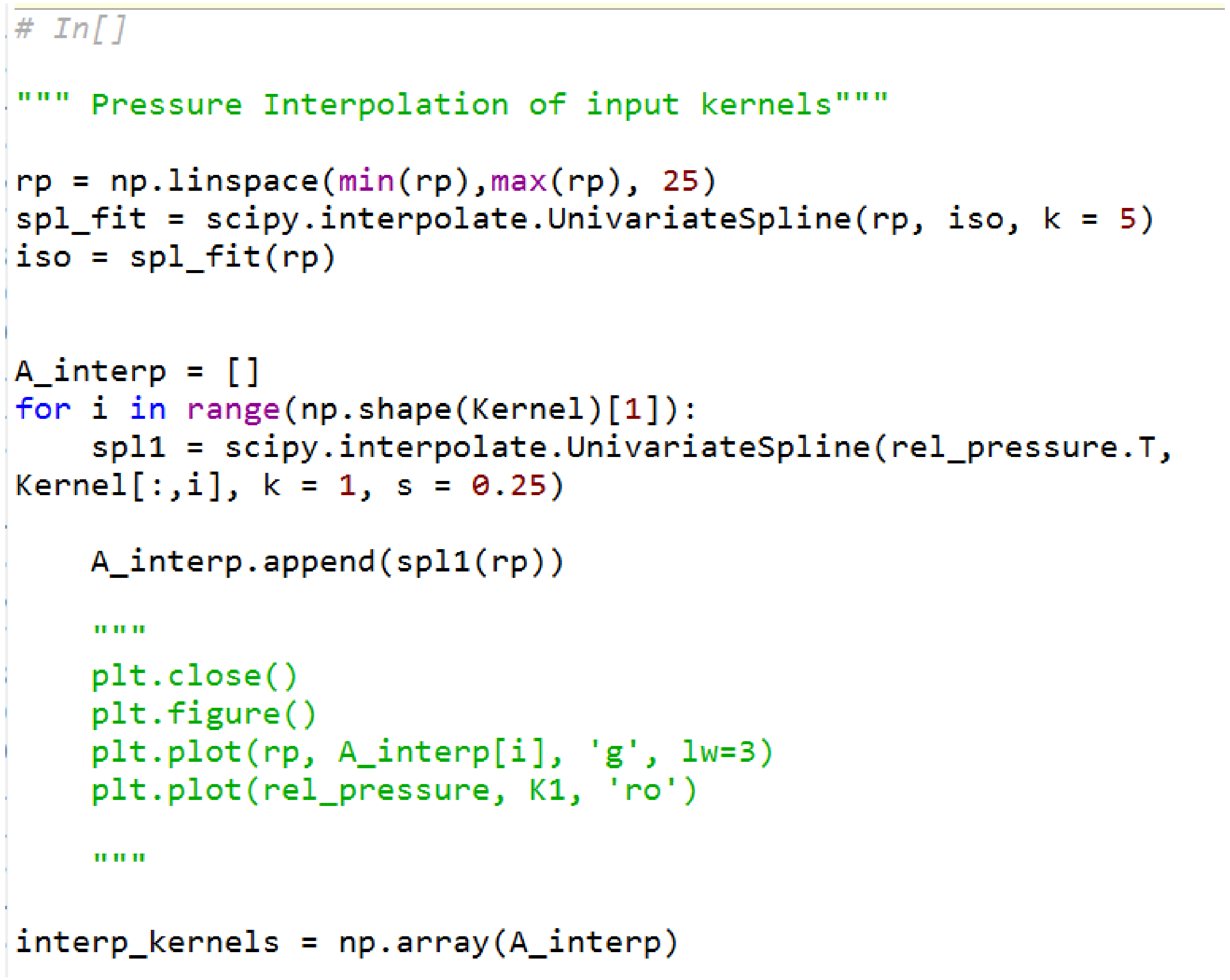
- To illustrate some of SciPy's capabilities, let's do the following:

  1. Data interpolation
     - Open the files named "GCMC_Kernels.txt" & "expt_isotherm.txt" as numpy arrays and parameterize columns
     - Generate kernels for the relative pressure values in the expt_isotherm.txt file
  2. Normal distribution fit
     - Generate an array using np.linspace()
     - Fit the array to normal distribution

Data Exploration and Modelling

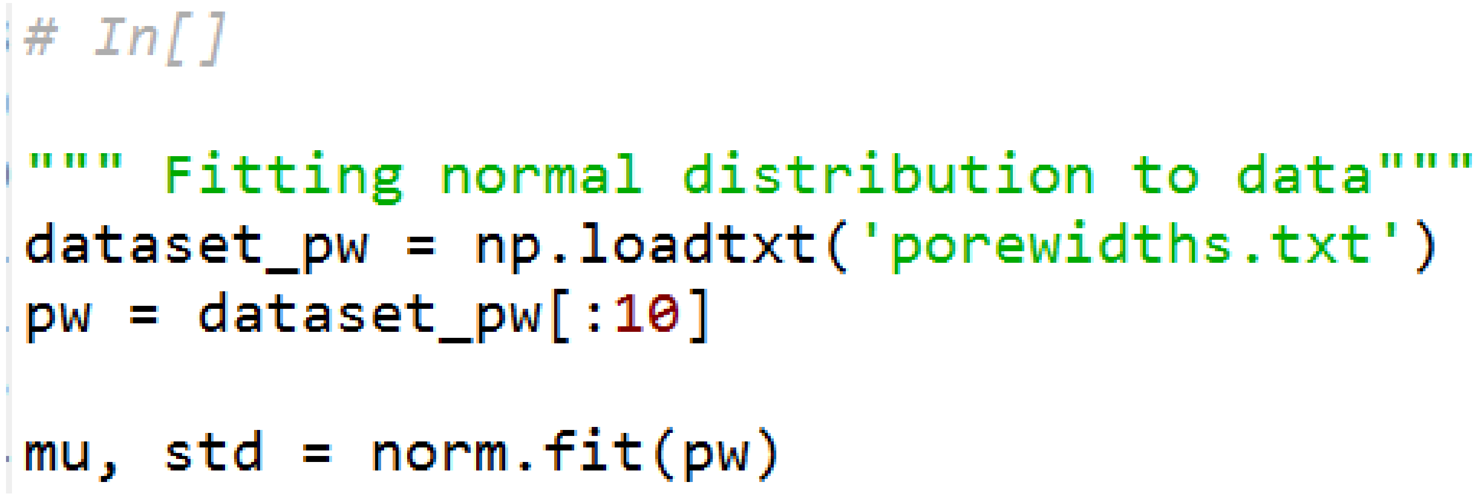
```
# In[]
import scipy
from scipy.stats import norm

""" Data Input & assignment"""

dataset = np.loadtxt('GCMC_Kernels.txt', skiprows = 1)
rel_pressure = dataset[:,0]
y = dataset[:,1:]
Kernel = dataset[:,1:11]


expt_data = np.loadtxt('expt_isotherm.txt')
rp = expt_data[:,0]
iso = expt_data[:,1]
```

Data Exploration and Modelling



```
# In[]

""" Pressure Interpolation of input kernels"""

rp = np.linspace(min(rp),max(rp), 25)
spl_fit = scipy.interpolate.UnivariateSpline(rp, iso, k = 5)
iso = spl_fit(rp)


A_interp = []
for i in range(np.shape(Kernel)[1]):
    spl1 = scipy.interpolate.UnivariateSpline(rel_pressure.T,
Kernel[:,i], k = 1, s = 0.25)

    A_interp.append(spl1(rp))

    """
    plt.close()
    plt.figure()
    plt.plot(rp, A_interp[i], 'g', lw=3)
    plt.plot(rel_pressure, K1, 'ro')

    """

interp_kernels = np.array(A_interp)
```

Data Exploration and Modelling



```
# In[]

""" Fitting normal distribution to data"""
dataset_pw = np.loadtxt('porewidths.txt')
pw = dataset_pw[:10]

mu, std = norm.fit(pw)
```

Session 4

Project 2

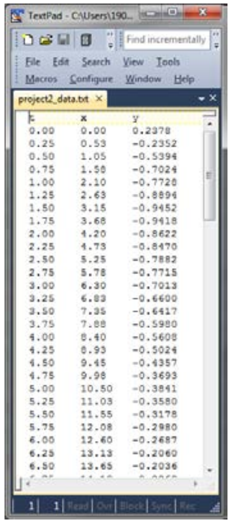
Project 2 – Curve-Fitting & Parameter Estimation

Aim: To apply the concepts learnt to a scientific dataset.

1. Apply python in-built functions (mostly string operations)
2. Define own functions (user-defined functions)
3. Visualize data and format plots

Description: Apply the curve-fitting module of SciPy to fit a dataset to a given model and estimate the optimal values & standard deviations of the fitting parameters

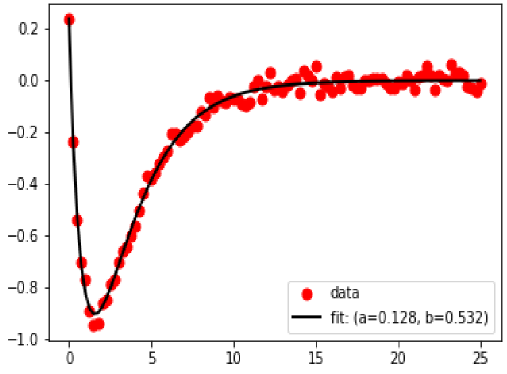
Input file: project2_data.txt


TextPad - C:\Users\190...
Find incrementally
File Edit Search View Tools
Macros Configure Window Help
project2_data.txt
1 1 Read Ovr Block Sync Rec

```
t       x       y
0.00    0.00    0.2378
0.25    0.53    -0.2352
0.50    1.05    -0.5394
0.75    1.58    -0.7024
1.00    2.10    -0.7728
1.25    2.63    -0.8894
1.50    3.15    -0.9452
1.75    3.68    -0.9418
2.00    4.20    -0.8622
2.25    4.73    -0.8470
2.50    5.25    -0.7882
2.75    5.78    -0.7715
3.00    6.30    -0.7013
3.25    6.83    -0.6600
3.50    7.35    -0.6417
3.75    7.88    -0.5980
4.00    8.40    -0.5608
4.25    8.93    -0.5024
4.50    9.45    -0.4357
4.75    9.98    -0.3693
5.00    10.50   -0.3841
5.25    11.03   -0.3580
5.50    11.55   -0.3178
5.75    12.08   -0.2980
6.00    12.60   -0.2687
6.25    13.13   -0.2060
6.50    13.65   -0.2036
```

+ **model = func(xdata, a, b)** →

Model parameters



References

- https://www.w3schools.com/python/default.asp
- https://docs.scipy.org/doc/scipy-0.18.1/reference/
- https://www.learnpython.org/en/Basic_String_Operations
- https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html
- https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html
- https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.array-creation.html
- Using Python for Research (HarvardX course)