Chapter 1 - Python Fundamentals

1 Python Fundamentals

Python is an interpreted, interactive, object-oriented, high-level programming language.

To break things down:

Interpreted – Any code we write in any programming language cannot directly be used by the computer. That is due to the fact that computers only work on electrical circuits and there are many intricate systems in place for it to function. For the computer to work, we need to provide it something it can understand (Machine Language) which is not our code. To translate to machine language, there are 2 ways to achieve it – Compiled or Interpreted. Compiled languages will translate the code into an executable program which is in machine language. This process is handled by a compiler thus the given name. Python on the other hand is not compiled but translated in run-time. An interpreter is used to achieve this feature and we will be working with the Python interpreter soon enough.

High-level – You will be working with a language with some semblance to human languages. In that regard, rather than working directly on computer components (such as registers, memory addresses etc.), the language provides higher level constructs to hide many of the complexities which makes it easier for developers to work with and learn.

Interactive – Due to the way the interpreter works, you can work directly on the Python prompt to write your programs and get results while you code. We will revisit this later on in this exercise

Object-oriented – This is a style of programming where we program in terms of objects. We will cover more on this later on in the course.

To use Python, an environment and generally an Integrated Development Environment (IDE) needs to be setup on your computer. The setup procedures are documented in another document named "Lab Setup". It is advised that the Python environment be ready before carrying on.

1.1 Python Programs

Think of the computer programs that we use in our everyday life. They range from big programs like software for server farms to medium programs like Microsoft Word to small programs like echoing a line of text to the terminal window. All these programs utilizes a common entity and that is a programming language is used to develop them. Learning programming is likened to learning a whole new language as there are numerous rules, idioms, jargon, patterns and paradigms to comprehend but let's take it one step a time.

Let's start off with a small Python program that prints a line of text to the screen.

```
1 # Printing 'Hello World' to the screen
2 print("Hello World!")
```

Running the above line of code in a Notebook code cell or the Python's interactive shell will produce the following output.

```
1 Hello World!
```

The command that we use for displaying the text to screen is called the `print()` function. This `print()` function is what we called a built-in function and it is provided to us when we installed the Python Programming Language into our computer. Python has several build-in components and libraries available for us to use out-of-the-box. We will be using some of the built-in libraries and others from 3rd party libraries for this course.

But before that, let's take a look at a slightly bigger program that uses data from a CSV file and plots a horizontal bar chart from small subset of the data available from the CSV file.

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  df = pd.read_csv('gst-by-economic-sector.csv', header=0, dtype='object')
5
6  mapping = {'na': 0}
7  df = df.replace({
8      'percentage_of_businesses_in_net_gst_refund_position': mapping,
9      'net_gst_contribution':mapping})
10
11 df = df.astype({'financial_year':int, 'no_of_businesses': int,
12             'percentage_of_businesses_in_net_gst_refund_position': float,
13             'net_gst_contribution': int, 'economic_sector': str,
14             'percentage_of_net_gst_contribution': float})
15
16 fig = plt.figure(figsize=(15,10))
17 temp = df[df['financial_year'] == 2011]
18 plot_axes = temp.plot.barh(x='economic_sector',
19                            y='percentage_of_net_gst_contribution')
20 plot_axes.get_legend().remove()
21 plot_axes.title.set_text("Year 2011")
22 labels = list(temp['economic_sector'].str.wrap(30))
23 plot_axes.set_yticklabels(labels)
24 plt.show()
```

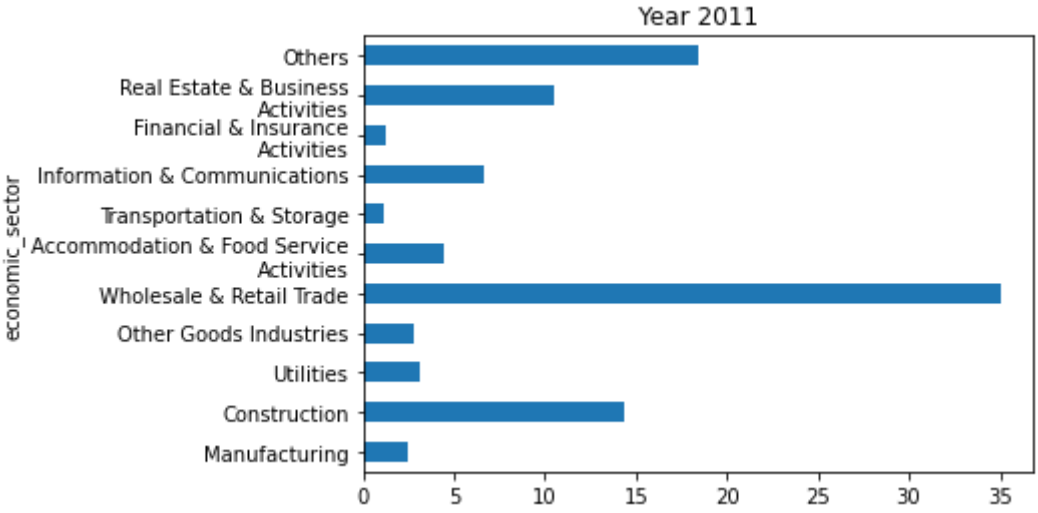

Figure 1: A bar chart generated from the code above.

This program seems to be overwhelming at the start but we will be learning how to do this and more in this course. You may not know many of the terms used to produce this program but we will be learning them very soon. But let's see what is happening in each line:

1. Importing (to make the library available to this program) the library called `Pandas` and shortening it to `pd`.

2. Importing the library called `matplotlib.pyplot` and shortening it to `plt`.
3. A blank line to make the code more readable.
4. Using the `Pandas` library to read a CSV file then load its data and set all its data to type `object`. Within the CSV file, there is the header at the first line of text.
5. A blank line to make the code more readable.
6. Creating a Python `dictionary` called `mapping`.
7. Beginning of a Pandas `replace()` function that searches the loaded data for a particular value and replaces it with another value.
8. The name of the column with records that has values that needs replacement.
9. The name of another column with records that has values that needs replacement and that ends the `replace()` function.
10. A blank line to make the code more readable.
11. Beginning of a Pandas `astype()` function that changes the data type of all the records in a particular column. The first 2 columns are to be converted to integers.
12. The next column is converted to float
13. The next 2 columns is converted to integer and string
14. The last column is converted to float and that ends the `astype()` function
15. A blank line to make the code more readable.
16. Creating a `figure` with the size 15 by 10 inches using the `matplotlib` library.
17. Extracting a subset of the loaded data that has records within the `financial_year` of `2011` and storing it in a variable called `temp`.
18. Starting of the `barh()` function from the Pandas object to plot a horizontal bar chart with the x-axis using data from the column `economic_sector` and saving the returned axes in a variable
19. Setting the y-axis of the bar chart to use data from the column `percentage_of_net_gst_contribution` and ends the `barh()` function
20. Using the axes to variable to tell `matplotlib` that we do not need a legend for the graph.
21. Using the axes to variable to set the title of the graph.
22. Creating a Python `list` to store a text wrapped (wrap after 30 characters) version of the text from the column *economic sector*.
23. Using the axes to variable to set the y-axis labels.
24. Show the plot to screen.

From the line by line explanation, we can see that libraries are rather useful in making short work of tasks that would be harder to develop using pure Python codes. Python programs are typically executed from the command line but we will also be exploring other methods that are commonly used to execute Python programs in the lab sessions. But before we dive in into the libraries, let's start with the basics.

1.2 Basics

From the Python programs in the previous section, we have seen that there are many different parts to a Python program such as variables, functions, libraries and objects. We will be going through all of these in due time in this course but let's start with the guidelines that Python developers follow when development Python programs. This guideline is called "Coding Style Guide". Normally within each organization there would be a document stating how the codes should appear and this is called "Style Guide" and within it, it would dictate the way the variables are named, the capital case of names, style of indention, etc. Style guides are used to aid consistency within development projects so that errors can be made obvious and future maintenance would be easier.

For this course we will be following the style guide outlined by Python here. Some of the more common points are:

- Variable names **must begin** with an underscore/s or a letter. Numbers not allowed. The remainder of the variable name can consist of letters, numbers or underscores
- Variable names are **case sensitive**
- Use **descriptive** naming styles
- Class names should use **CapWords convention**
- Function names should be in lowercase, with words separated by underscores as necessary to improve readability.

With Python 3, it is also possible for variable and function names to support non-ASCII identifiers. That means some languages that has accented characters, Cyrillic, Greek, Kanji, etc are now possible to be used as identifiers. However the support only extends to UTF-8 representation of those letters and emojis are not allowed. A note of caution when using non-ASCII characters for variable and function names, if the code is to be shared, it might be difficult for others to read and understand it if they are not familiar with the symbols or native language thus producing illegible codes.

Some examples of Python syntax following Python's Style Guide:

```
# variables
int_var = 10
_float_var = 5.2      # protected variable
__str_var = "hello"   # private variable
dog = Animal()        # user defined object
π = 3.14159           # Greek alphabet
ラーメン = "ramen"       # Japanese Kanji

# functions
def __init__(self):
def calculate_interest():
self.cal_compound_interest()

# classes
class Node:
class Animal:
```

1.3 Commenting in Python

Commenting codes in programming is very important as it provides a human readable explanation of what the codes or function are used for. Comments are generally ignored by compilers and interpreters provided that the right comment syntax is used. For Python, comments are done using the hash symbol (`#`) for single line comments or enclosed in triple single or double quotes (`"""` or `'''`) for multiline comments.

```
# this is a single line comment

'''
This is a multiline
comment.
'''

"""
This is also a multiline
comment.
"""
```

Comments can be added almost any where within the code and it is advisable to add comments to your code so that you can use it for future reference.

In addition to commenting, there will be times where a line of code is too long and we need a way to break it up into multiple lines for ease of reading. To do this, we use the backslash character `\` to tell the interpreter that the line has not concluded or we can also break a line up at the commas.

```
# for breaking up arithmetic equations
result = 1 + 5 + 92 + 14 + 64 + \
         5**2 + 9**3 + 2*2

# for functions
def foo_bar(input_1, input_2, input_3,
            input_4, input_5)
```

1.4 Requesting User Input

In almost all software programs that most of us have used, we would have come across some form of user input. User input is an integral part of most (if not all) software that help its users complete a task. Python is no different, we will be learning to create programs and some of those programs will require some form of user input. To request for user input, we use the function `input(<prompt>)` where `<prompt>` is the message shown to the user to indicate that their input is required. An example is as follows

```
user_input = input("Enter a letter then press enter.")
```

Pressing the `Enter` key on the keyboard after entering a value will "tell" the `input()` function that the user has finished entering their inputs. The user's input is then stored into the variable called `user_input` using the `=` assignment operator. Note that when the `input()` function is used, the program will pause its execution until the `Enter` key on the keyboard is pressed regardless of whether or not the user has entered a value.

1.5 References

1. Python 3 - Basic Syntax, https://www.tutorialspoint.com/python3/python_basic_syntax.htm
2. PEP 3131 -- Supporting Non-ASCII Identifiers, https://www.python.org/dev/peps/pep-3131/