ET0735 - DEVOPS FOR AIOT
SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING, SINGAPORE POLYTECHNIC

LABORATORY 2: INTRODUCTION TO PYTHON

Objectives

By the end of the laboratory, students will be able to

- Implement a basic Python program using the following,
  - Logical Operators
  - Mathematical Operators
  - String processing
  - Data processing

Activities

- Installation of Python 3 interpreter
- Installation of PyCharm Professional Edition using "Educational License"
- Create a new Python project in PyCharm
- Create and Execute Python scripts in PyCharm project
- Commit and Push to GitHub
- Basic Python programming: Functions, console Input / Output, strings and list data structures, comparison and arithmetic operators.

Review

- Successfully installed PyCharm Professional
- Created a Python application using basic logical, mathematical operators with string and data processing
- Implement basic Python console application to get user input from the terminal console

Equipment:
Windows OS laptop

Procedures:

1 Installation of Python 3 Interpreter

Before we start to install the PyCharm Professional IDE, we first need to install the Python 3 interpreter which PyCharm will use for running our Python code.

1.1 Download the Python 3 interpreter for Windows from the link below. Choose version 3.8.x or later (e.g. 3.8.10 for 64-bit systems)

https://www.python.org/downloads/

1.2 Run the installer to start the Python 3 interpreter installation process. During installation, remember to tick "**Add Python 3.8 to PATH**" and "**Disable path length limit**".

2 Installation of PyCharm Professional Edition using "Educational License"

2.1 Download PyCharm version 2021.3.3 or higher from the following link below. Use you SP "iChat" email address to register. Choose **PyCharm "Professional Edition"**. You will use "**educational license**" at no cost for 1 year.

https://www.jetbrains.com/shop/eform/students

2.2 Install the PyCharm "Professional Edition" using the educational license using your SP "iChat" email address. During the installation, tick the 4 options:

- Create Desktop shortcut
- Update PATH variable
- Update Context Menu
- Create associations

3 Create a new PyCharm project

3.1 In your laptop's C-drive, create a new folder in the Local_Git_Repository folder that you have created in Lab1 "c:\Local_Git_Repository\Lab2". You will use it to

store your Python codes. Take note that this is just for example and you can create a folder for this lab in any preferred location in your laptop.

3.2 Launch the PyCharm software by double-clicking the shortcut on Desktop. A welcome window will pop up. Choose "New Project".

3.3 The "Create Project" dialog box will appear. Make sure that "Pure Python" has been selected. (Note: the colour of your popup window may be different from what is shown in Figure 1 due to different theme chosen for your PyCharm software.)

3.4 For the "Location" field, click the folder icon at the far right. Choose the "c:\Local_Git_Repository\Lab2" folder you had created in Step 3.1, then click "OK". This will set the location for storing your Python project.

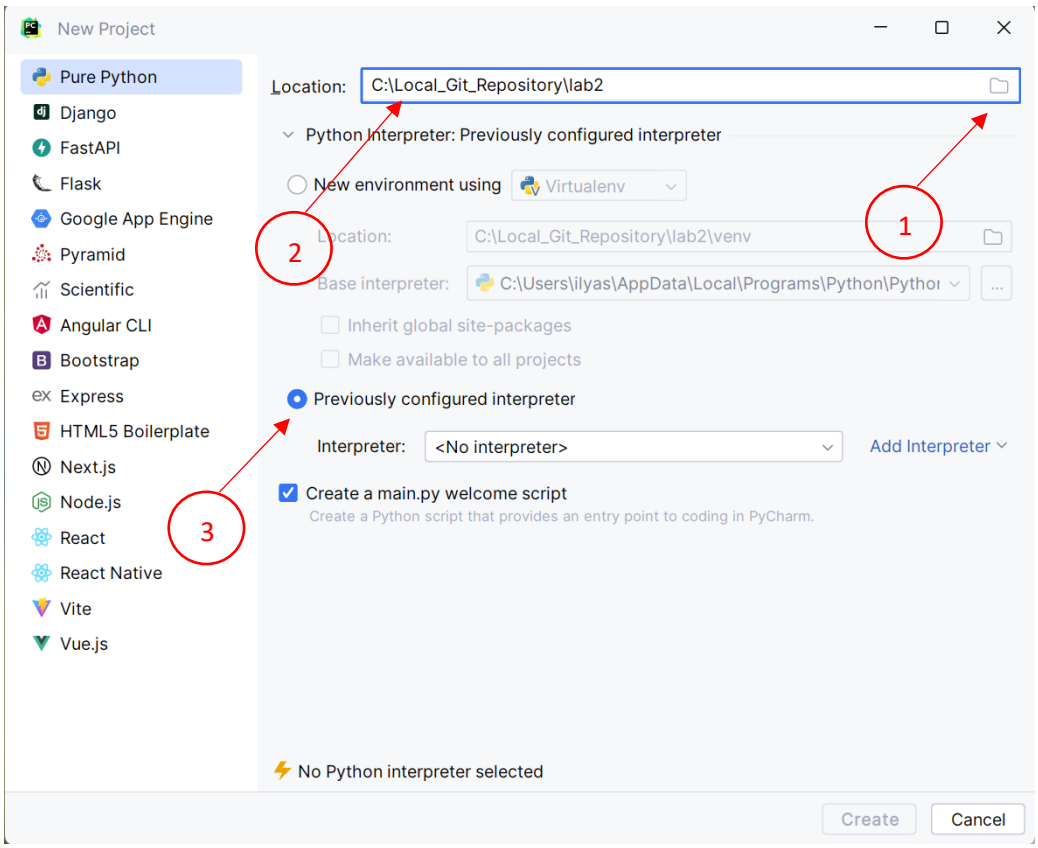

Figure 1 – Set the saving location of your Python project.

3.5 Click on the “previously configured interpreter”

3.6 Since there is no interpreter available, we need to select <Add interpreter> option, and then select “Add local interpreter” (Figure 2)

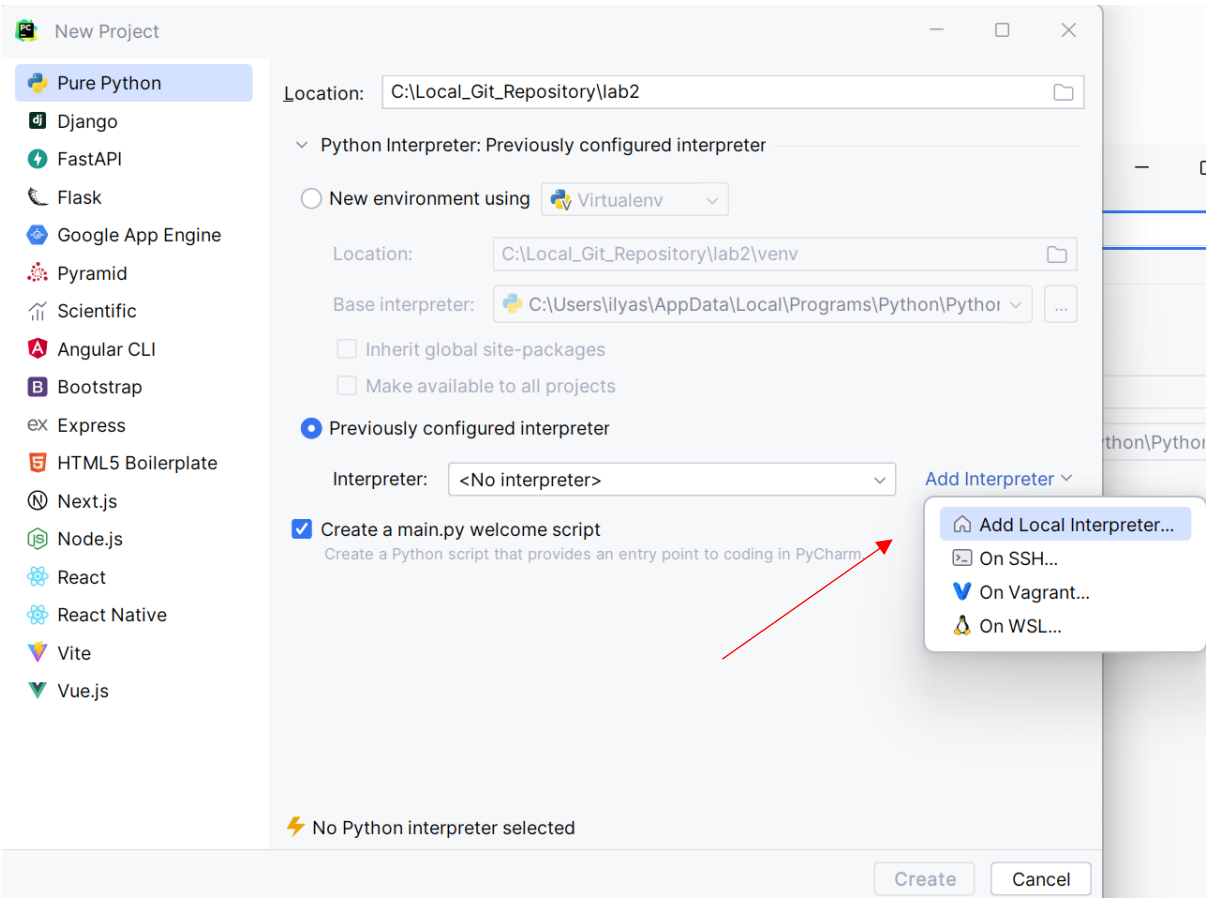

Figure 2 – Add Local Interpreter in PyCharm

3.7 In the “Add interpreter” Select system interpreter and the system should select the python version that you have installed in your laptop (Figure 3).

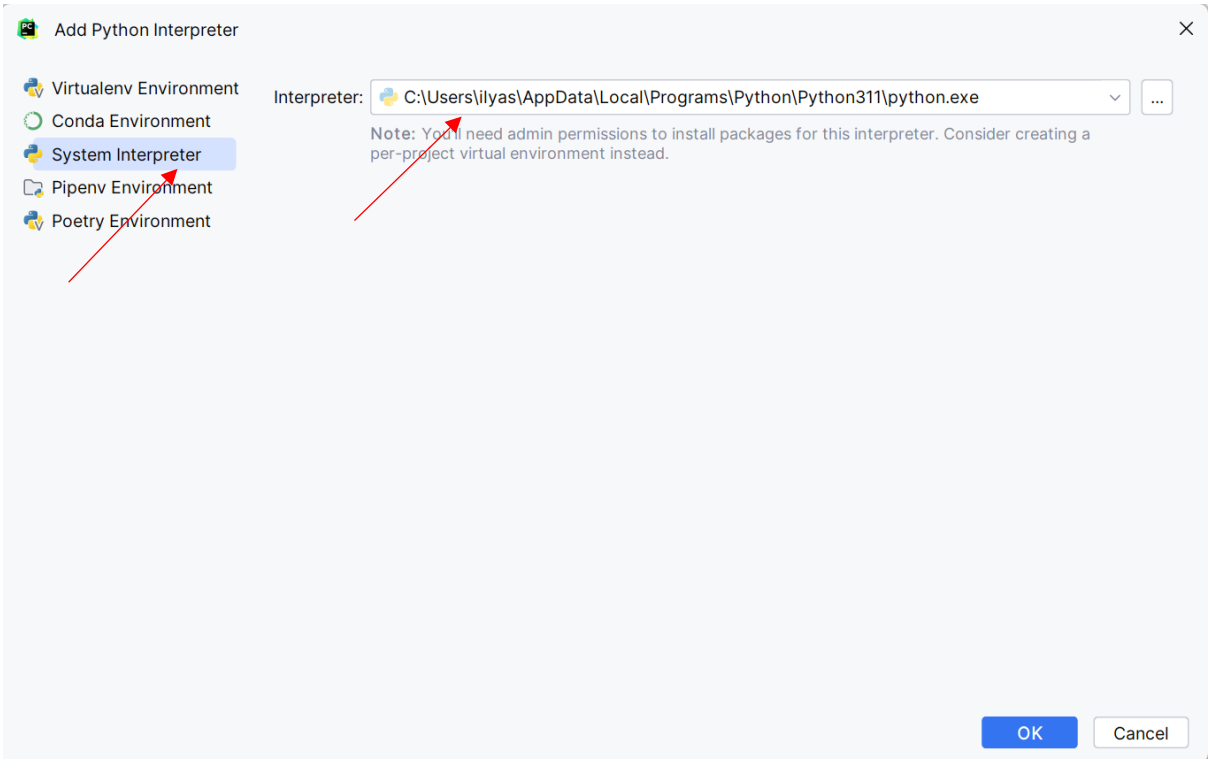

Figure 3 – Choose the interpreter for your Python project.

3.8 The PyCharm IDE will be launched, showing the newly created project Lab2.

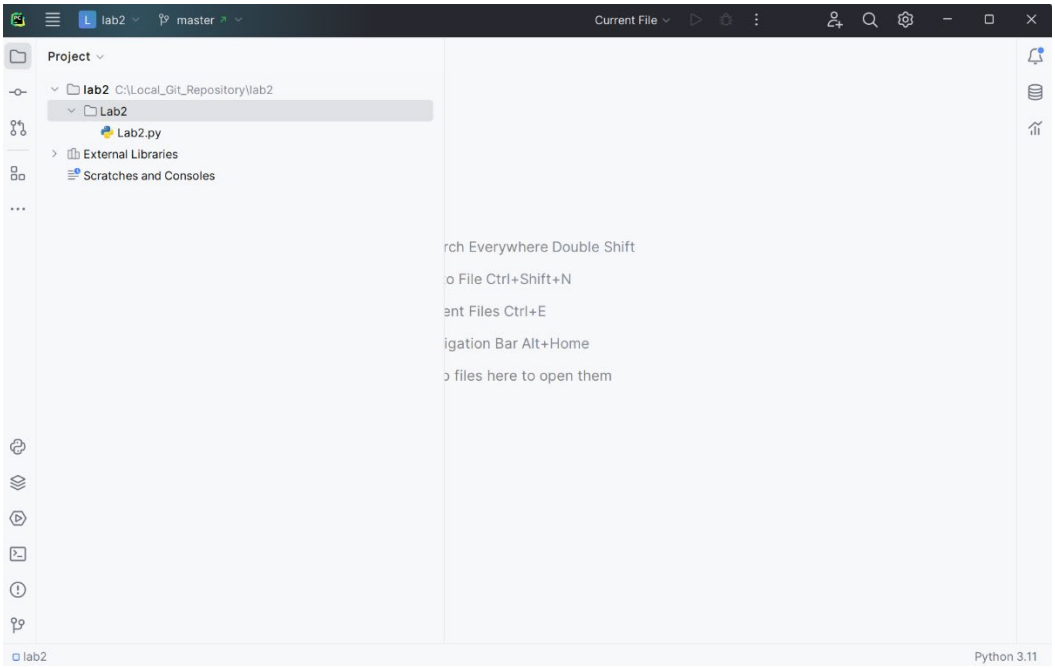

Figure 4 – The PyCharm IDE.

3.9 While in the PyCharm IDE, you may click "Help→About" to see your PyCharm software licensing information. **Check that you have correctly installed PyCharm "Professional Edition" and the expiry date is at least 1 year from the current date.**

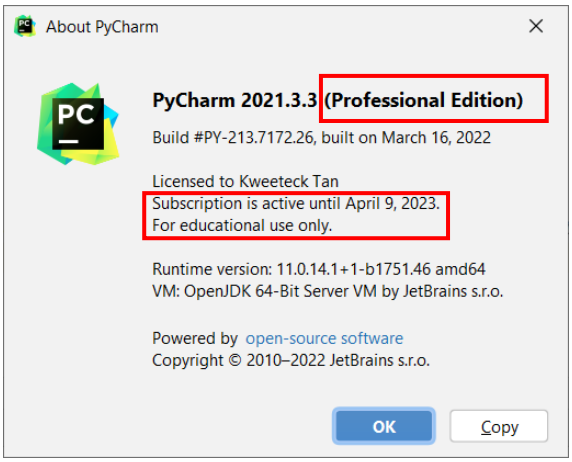

Figure 5 – Use "educational license" for the PyCharm (Professional Edition) software.

4 Create and Execute Python scripts in a PyCharm project

4.1 In PyCharm IDE, right-click the "ET0735_Lab2" project, choose "New", and then select "Python File".

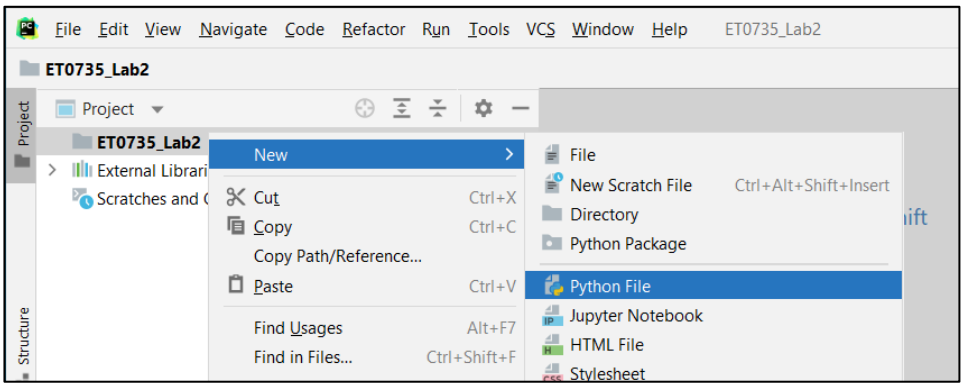

Figure 6 – Add a new Python file to your PyCharm project "Lab2".

4.2 A popup window will prompt for the file name. Name the file "Lab2.py" and press the ENTER key.

4.3 Add the the following Python code in the newly created Lab2.py,

```
print("ET0735 (DevOps for AIoT) - Lab 2 - Introduction to Python")
```

4.4 From the "Run" menu, select "Run…" to run Lab2.py code.

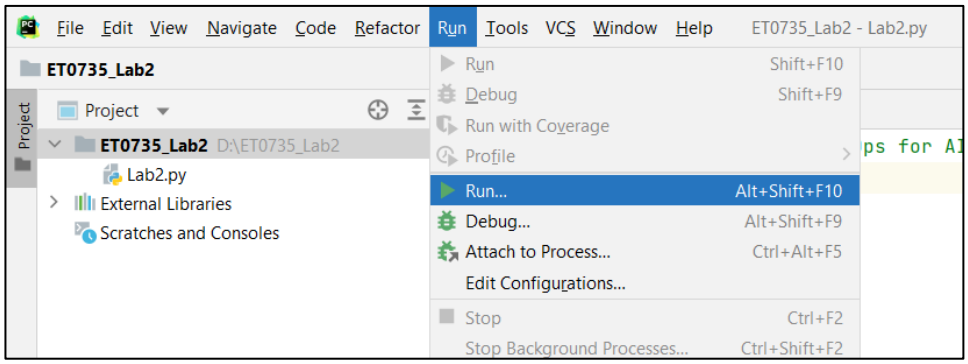

Figure 7 – Run the Lap2.py code.

The first time the "Run" option is selected in PyCharm, a dialog box will appear for you to select which Python script should be run and you should select "Lab2.py".

Subsequently the "Run" option will execute the last run Python script without prompting you to select which script to execute.

4.5 Check the console output in PyCharm.

If your project was created correctly you should see the console output based on the initial code in Lab2.py.

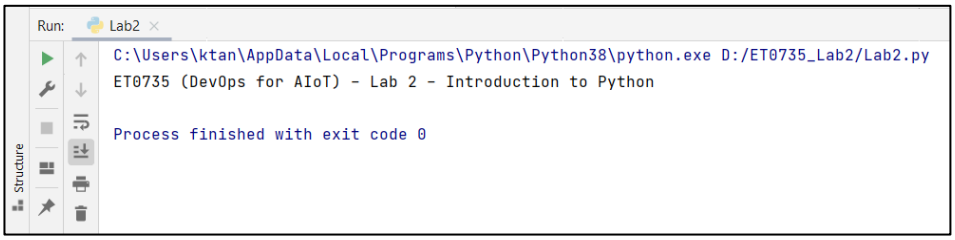

Figure 8 – Console output after running the Lab2.py code.

5 Create a Local Git Repository and Push to GitHub

5.1 Now, you will create a new local git repository to track your PyCharm Python project.

5.2 Open a Command Prompt window and change to the directory to "c:\Local_Git_Repository\Lab2".

5.3 Enter Git command **`git init`** and check that the .git folder has been created in the "c:\Local_Git_Repository\Lab2"

5.4 Stage all the contents in the folder "c:\Local_Git_Repository\Lab2" using the **`git add *`** command. The '*' represents a wildcard filter that requests git to stage all files within the current repository folder.

5.5 After the staging is completed, commit to the local git repository. Write the git command you use in the space provided below.

```
git commit -m "commiting lab2"
```

5.6 Login to GitHub using the account you had created in Lab 1. **Create a new repository at GitHub and name it as Lab2.** Copy the URL of the GitHub Lab2 repository. Refer to Lab 1 for the steps if necessary.

5.7 At your local git repository, specify the URL of the remote GitHub repository. Set up the upstream branch. After that, push the local git repository to GitHub. Write the git commands you use in the space provided below.

```
git remote add origin https://github.com/PatrickWendell/Lab-2.git
  git remote -v
git push --set-upstream origin master
```

5.8 Notice that not all the files located in the folder "c:\Local_Git_Repository\Lab2" are pushed to GitHub. To understand which files are ignored and not tracked in the local and remote repository, open the file .gitignore in "D:\ET0735_Lab2\.idea" and study it's contents to understand which files will be "ignored" by Git.

6 Python Functions and Mathematical Operators

Python User Defined Functions

To develop modular software code that is easy to maintain and extend, we need to decompose our Python code into functions.

Splitting the code into functions are also useful to simplify the coding implementation as well as to simplify the Software Unit Testing which will be covered in Lab 3.

Python is an indentation-based (each indentation is 4 spaces) programming language and functions are defined with the following syntax:

```
def funcName(parameter1, parameter2):
    print("this is a dummy function")
    return 10
```

Notice in the above code we define a function starting with the keyword "def" followed by the function name and then a list of comma separated parameters. The code inside the function is indicated to start with a tab space (= 4 spaces), and the function can return some data using the "return" keyword.

Python Arithmetic Operators

The table below list some of the most commonly used Python arithmetic operators.

Mathematical Operator	**Python Operator**	**Example**
Addition	+	result = x + y
Subtraction	-	result = x - y
Multiplication	*	result = x * y
Division	/	result = x / y
Exponential	**	result = x ** y
Modulo	%	result = x % y

Table 1

For further information on Python arithmetic operators, refer to the URL below,

https://www.w3schools.com/python/gloss_python_arithmetic_operators.asp

Python Conditional Operators

Please refer to the link below for examples of Python conditional operators.

https://www.w3schools.com/python/python_conditions.asp

Exercise 1:

Implement a Python application that will calculate the Body Mass Index (BMI) based on the user's height (in meters) and weight (in kilograms).

(a) In PyCharm, using the project "**Lab2**" created in step 3, create a new Python file "bmi.py" and define a new function "calculate_bmi" with 2 string parameters for height and weight as shown below

```
def calculate_bmi(height, weight):
    print("Height = " + height)
    print("Weight = " + weight)

calculate_bmi(weight="57", height="1.73")
```

Run the Python code above and observe the console output in PyCharm.

(b) Based on the Python code implemented in (a), modify the code to call the function "calculate_bmi" using floating point values instead of strings. **Notice the difference in the weight and height parameters now passed as floating point data type without the quotation marks "".**

```
def calculate_bmi(height, weight):
    print("Height = " + height)
    print("Weight = " + weight)

calculate_bmi(weight=57, height=1.73)
```

Run the Python code with the modifications in bold above and observe the console output which should now result in a runtime error.

Write down the last error observed in PyCharm console in the box below

```
TypeError: can only concatenate str (not "float") to str
```

Modify the function "calculate_bmi()" to use the str() function as shown in the code below.

```
def calculate_bmi(height, weight):
    print("Height = " + str(height))
    print("Weight = " + str(weight))

calculate_bmi(weight=57, height=1.73)
```

Run the modified Python code passing the weight and height as floating point data types and check the PyCharm console ouput.

In the box below, write down the reason why the “str()” function has helped to fix the error.

any number initialize with height and weight will become string without needing a ""

(c) Finally we now need to implement the mathematical formula to calculate BMI based on the formula below,

$$\boldsymbol{BMI} = \frac{\boldsymbol{Weight\ (kg)}}{\boldsymbol{Height\ (m)} \times \boldsymbol{Height\ (m)}}$$

```
def calculate_bmi(height, weight):
    print("Height = " + str(height))
    print("Weight = " + str(weight))

    #Add code here to calculate BMI
    bmi = ..

    #Add code here to display calculate BMI
    print(. . .)

calculate_bmi(weight=57, height=1.73)
```

Modify the code above to implement the Python code to calculate and display the BMI value on the console.

Run your Python file and check that the PyCharm console shows the correct calculated BMI value.

(d) Based on the table below, implement Python code using conditional operators (eg: if, else, etc) to determine if the user is “Under Weight”, “Normal Weight” or “Over Weight”.

BMI Range	Weight Classification
BMI < 18.5	Under Weight
18.5 ≤ BMI ≤ 25.0	Normal Weight
BMI > 25.0	Over Weight

Table 2

Update your Python code to display on the console the BMI classification and verify that the results are correct based on the BMI ranges defined in Table 2.

(e) Commit and Push all your local Git repository changes to Github

(f) Create a new **Git tag "Lab2_v1.0"** and push the tag to Github.

7 Python Main Entry Point, Console Input and String Processing

Python Main Entry Point

In the previous exercise, we have implemented Python code that calculates the BMI value using a user defined function "calculate_bmi".

Also notice that we called the function "calculate_bmi()" directly instead of a central "main" function or entry point.

While this works for a simple single Python file application, calling Python functions directly from a file might become messy if the application comprises of several Python files.

In this case it would be better to define a single Python file and "main" function as an entry point to start the entire application.

We should define the **main entry point** function "main()" which can then be used to call other user defined functions.

```
def main():
    print("ET0735 (DevOps for AIoT) - Lab 2 - Introduction to Python")
    display_main_menu()
    num_list = get_user_input()
```

In Python, the code below checks if the current script is the main Python file and then calls the function "main()" that we have defined in the step above.

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

Exercise 2:

7.1 For this exercise, we will develop a console application that allows the user to enter several numeric values and the Python code will calculate the average, minimum and maximum values for the list of temperature values.

7.2 In the Python file Lab2.py create the functions shown in Table 1 below, with only a single "print" statement to display the name of the function. The actual implementation of the functions in Table 1 will be completed in the next sections.

Example 1: For the function "display_main_menu()"

```
def display_main_menu():
    print("display_main_menu")
```

Example 2: For the function "calc_average()",

```
def calc_average():
    print("calc_average")
```

Function Name	**Input Parameter/s Type**	**Return Type**
display_main_menu	None	None
get_user_input	None	List of Floats
calc_average	List	Float
find_min_max	List	List of Floats in the format [min_temp, max_temp]
sort_temperature	List	List of Floats sorted in ascending order
calc_median_temperature	List	Float

Table 3 – Functions to be added to Lab2.py file.

Console Input / Output, Strings and List Data Structures
Exercise 3:

7.3 Based on the functions skeletons created in Table 3, complete the Python code implementation as shown in Table 4. Refer to Table 3 for the expected input parameters and return type for each function.

Function name	Python code implementation
display_main_menu( )	To display the following text: *"Enter some numbers separated by commas (e.g. 5, 67, 32)"*
get_user_input()	1. Read from the terminal console a sequence of numbers entered by the user using the Python system function "input()" https://www.w3schools.com/python/ref_func_input.asp 2. Use the Python function "split(",")" to split the user entered string into a Python List of strings based on the ",' comma character. https://www.w3schools.com/python/ref_string_split.asp 3. Convert the Python List of strings to a List of floats which can be used later for calculating average, minimum and maximum values https://www.w3schools.com/python/python_lists.asp 4. Return the List of floats

Table 4 – Implementation of two functions in Lab2.py file.

Comparison and Arithmetic Operators

In this exercise, you will need to implement some mathematical operations to calculate the average, minimum and maximum values of the list of numbers entered by the user in the previous exercise.

Exercise 4:

7.4 Based on the functions created in Table 1, complete the Python code implementation as shown in Table 3. Refer to Table 1 for the expected input parameters and return type for each function.

Function name	Python code implementation
calc_average_temperature()	To return a float value of the calculated average value of all temperature readings.
calc_min_max_temperature()	To return an integer list with 2 values for minimum and maximum temperature.

Table 5 – Implementation of two more functions in Lab2.py file.

Commit, Push to GitHub and create Software Release

Exercise 5:

7.5 Add and Commit all changes to your local Git repository with appropriate commit messages that describe your code changes

7.6 Create a **Git tag "Lab2_v1.1"** in your local repository.

7.7 Push all committed changes in your local git repository to your remote GitHub repository.

7.8 Create a new **Github release "Lab2_v1.1"** based on the corresponding Git tag.

8 Additional Exercise 6:

8.1 Implement the function "calc_median_temperature()" defined in Table 1 which returns the median temperature from the list of numbers entered by the user. (Hint: Before deriving the median value, you will need to first sort all the numbers in the list in ascending order.)

8.2 Push all committed changes in your local repository to your remote GitHub repository

8.3 Create a new **GitHub release "Lab2_v1.2"** based on the corresponding Git tag.