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 Visual Studio Code
- Create a new Python project in Visual Studio Code
- Create and Execute Python scripts in VSCODE 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 Visual Studio Code
- 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 Visual Studio Code IDE, we first need to install the Python 3 interpreter which Visual Studio Code will use for running our Python code.

1.1. Download the Python 3 interpreter for Windows from the link below. Choose version 3.11.x or later (e.g. 3.11.10 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 3.11 to PATH**" and "**Disable path length limit**".

2. Installation of Visual Studio Code

2.1. Download Visual Studio Code (Stable Build) from the following link below.

https://code.visualstudio.com/

2.2. Install Visual Studio Code. During the installation, press next until you reach install. Tick "Create a Desktop Shortcut" to create a desktop icon.

3. Create a new VSCODE 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 VS Code by double-clicking the shortcut on Desktop. A welcome window will pop up.

3.3. Press the extensions tab on the left, and search for Python. Download the Python extension. (Figure 1)

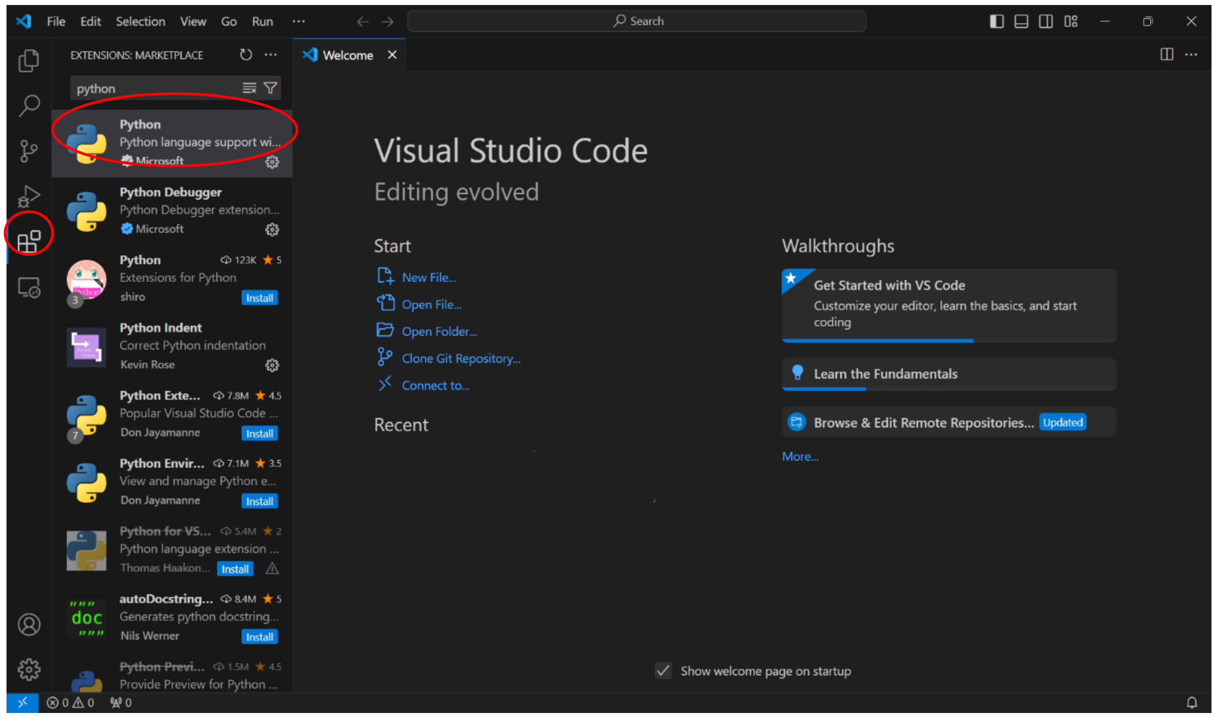


Figure 1 – Installing Python Extension

3.4. After downloading the Python extension. In the “Welcome” Screen, press on “Get started with Python Development”. Expand “Select or create a Python environment”, and click on “Select Python Interpreter”. After that click on Python 3.11 at the top. Then, press “Mark Done” below. (Figure 2)

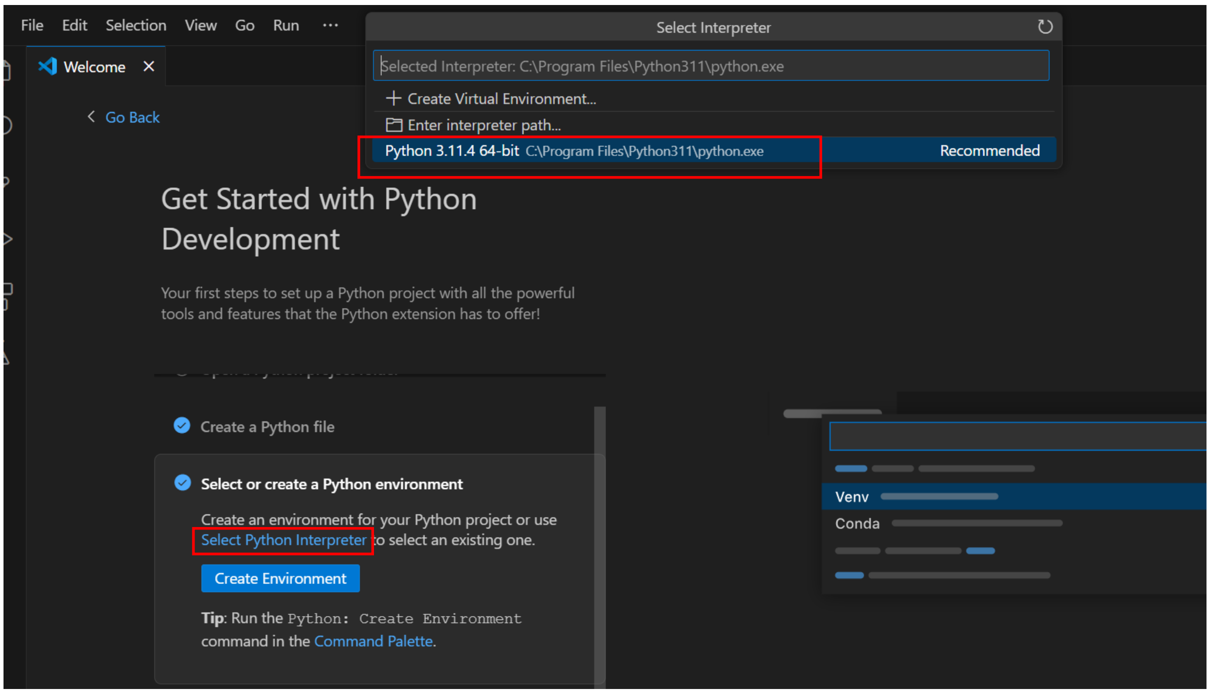

File
Edit
Selection
View
Go
Run
Welcome
Go Back
Select Interpreter
Selected Interpreter: C:\Program Files\Python311\python.exe
Create Virtual Environment...
Enter interpreter path...
Python 3.11.4 64-bit C:\Program Files\Python311\python.exe
Recommended
Get Started with Python Development
Your first steps to set up a Python project with all the powerful tools and features that the Python extension has to offer!
Create a Python file
Select or create a Python environment
Create an environment for your Python project or use Select Python Interpreter to select an existing one.
Create Environment
Tip: Run the Python: Create Environment command in the Command Palette.
Venv
Conda

Figure 2 – Selecting a Python Interpreter

3.5. Back at the Welcome Screen, at the explorer panel, press “Open Folder…” and navigate to the Lab2 folder that you created previously in Step **3.1**. (Windows (C:) → Local_Git_Repository → Lab2). Then, click Select Folder. This will open the Lab2 folder.

3.6. The Visual Studio Code IDE will be launched, showing the newly created project Lab2. (Figure 3)

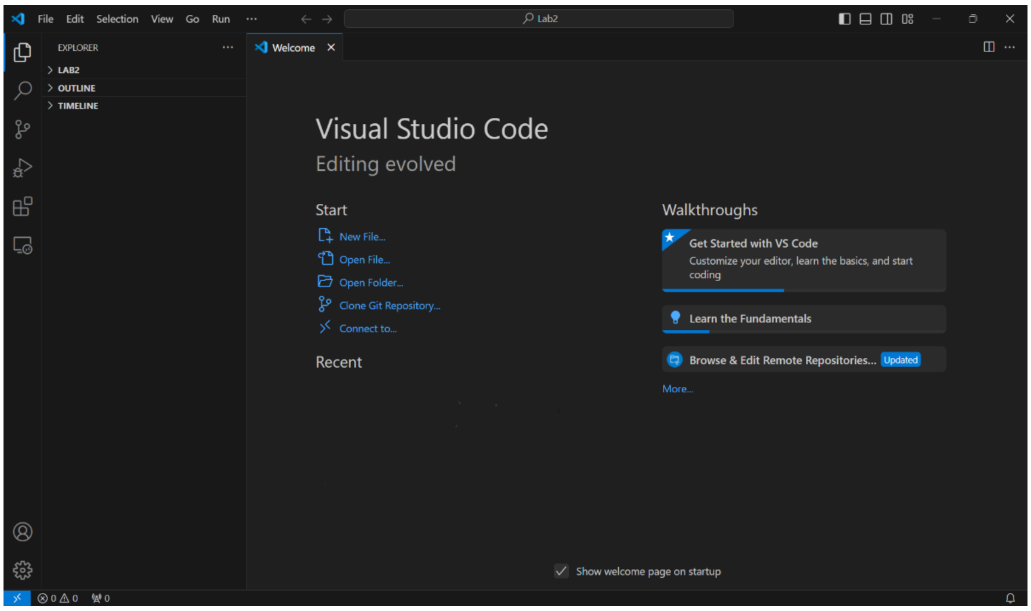


Figure 3 – The Visual Studio Code IDE

4. Create and Execute Python scripts in a VSCODE project

4.1. In VSCODE IDE, expand the "Lab2" project, choose "New File". (Figure 4)

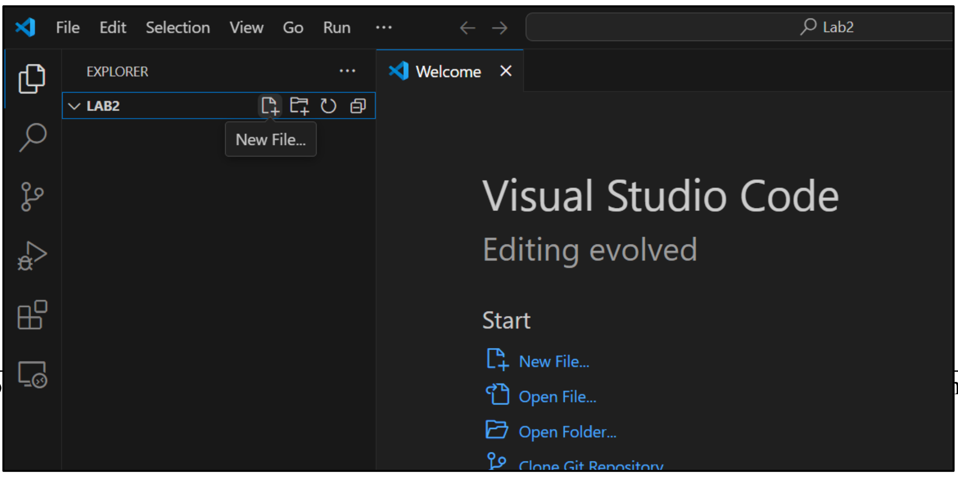


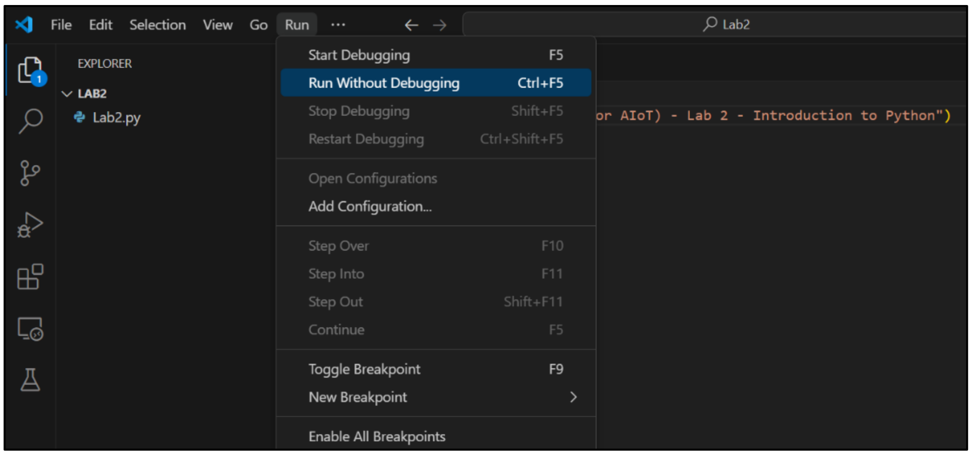
Figure 4 – Add a new Python file to your VSCODE project "Lab2".

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

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

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

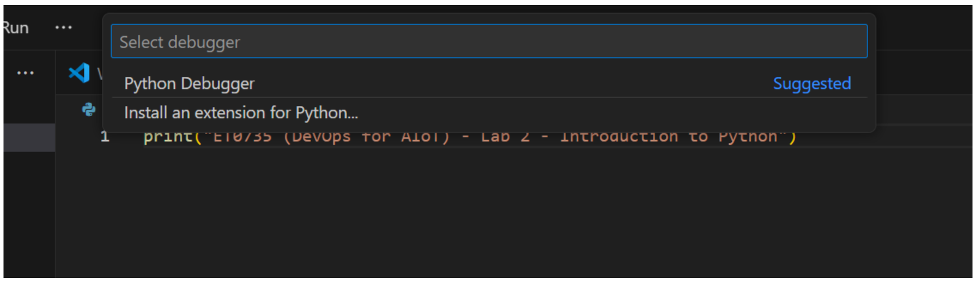
4.4. From the "Run" menu, select "Run Without Debugging" to run Lab2.py code.



```
or AIoT) - Lab 2 - Introduction to Python")
```

Figure 5 – Run the Lab2.py code.

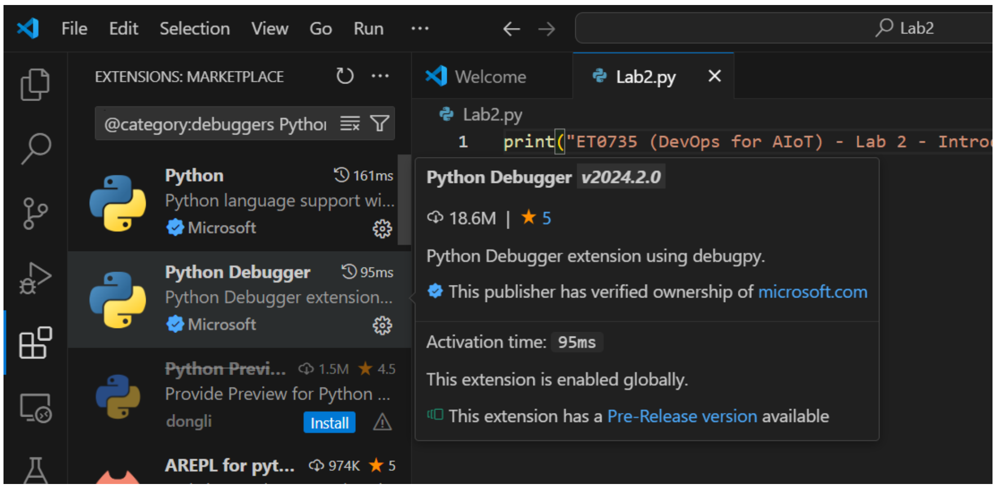
The first time the "Run Without Debugging" option is selected in VSCODE, a dialog box will appear for you to select a debugger. Select "Python Debugger".



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

Figure 6 – Select Python Debugger

If this option doesn't appear, you may have to install Python Debugger extension from the extensions marketplace.



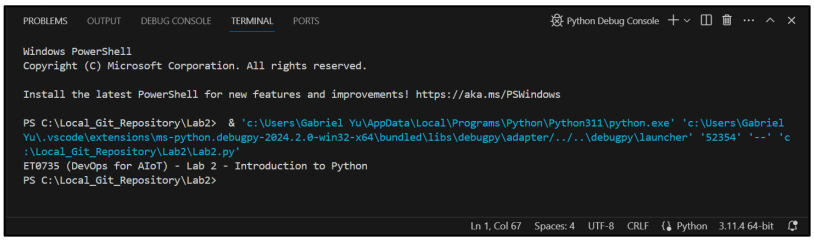
```
Lab2.py
print("ET0735 (DevOps for AIoT) - Lab 2 - Intro
```

Python Debugger v2024.2.0
18.6M | ★ 5
Python Debugger extension using debugpy.
This publisher has verified ownership of microsoft.com
Activation time: 95ms
This extension is enabled globally.
This extension has a Pre-Release version available

Figure 7 – Install Python Debugger

4.5. Check the console output in Visual Studio Code.

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



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Local_Git_Repository\Lab2>  & 'c:\Users\Gabriel Yu\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Gabriel
Yu\.vscode\extensions\ms-python.debugpy-2024.2.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '52354' '--' 'c
:\Local_Git_Repository\Lab2\Lab2.py'
ET0735 (DevOps for AIoT) - Lab 2 - Introduction to Python
PS C:\Local_Git_Repository\Lab2>
```

Ln 1, Col 67   Spaces: 4   UTF-8   CRLF   Python   3.11.4 64-bit

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 VSCODE 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. Use **git add .**

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 "Initial commit"
```

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.** Do not use "-" or start the repository name with a number. This is to prevent issues for using the lab2 as a module for your future lab sessions. 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 "yourRepoUrL.git"
```

```
git push -u origin main
```

To check whether you have successfully pushed:

1. **git branch ——v**

Alternatively, if you GitHub CLI

gh repo create Lab2 ——public

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 Visual Studio Code, 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 Visual Studio Code.

(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 Visual Studio Code 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 Visual Studio Code console ouput.

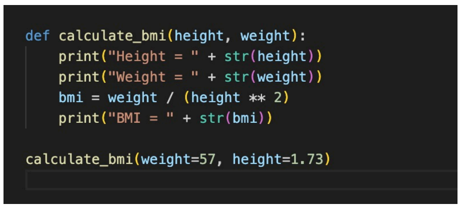
In the box below, write down the reason why the “str()” function has helped to fix the error.

str( ) converts the numeric values (float) to strings, enabling string concatenation with the + operator. Python requires both operands to be the same type for concatenation.

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

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

```
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)
```



```
def calculate_bmi(height, weight):
    print("Height = " + str(height))
    print("Weight = " + str(weight))
    bmi = weight / (height ** 2)
    print("BMI = " + str(bmi))

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.

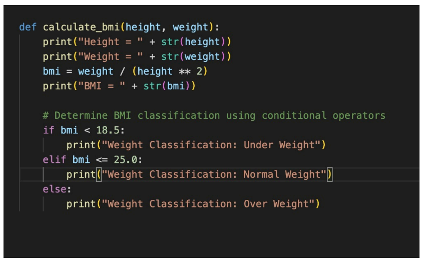
Run your Python file and check that the Visual Studio Code console shows the correct calculated BMI value.

(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.



```
def calculate_bmi(height, weight):
    print("Height = " + str(height))
    print("Weight = " + str(weight))
    bmi = weight / (height ** 2)
    print("BMI = " + str(bmi))

    # Determine BMI classification using conditional operators
    if bmi < 18.5:
        print("Weight Classification: Under Weight")
    elif bmi <= 25.0:
        print("Weight Classification: Normal Weight")
    else:
        print("Weight Classification: Over Weight")
```

(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.

To push a specific tag:

git tag -a Lab2_v1.0 -m “Completed Exercise 1: BMI Calculator”

git push origin Lab2_v1.0

Why tags?
Tags are like permanent bookmarks in your Git history that says “this specific commit is important”
By default tags apply to the latest commit, to tag a specific older commit **git tag -a test -m “test’ 8ff507e**

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

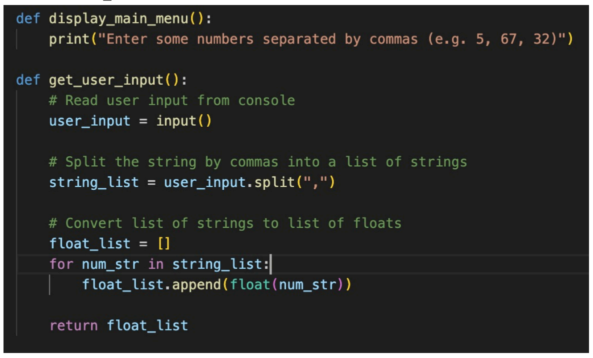
Table 3 – Functions to be added to Lab2.py file.

<u>Console Input / Output, Strings and List Data Structures</u>
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.



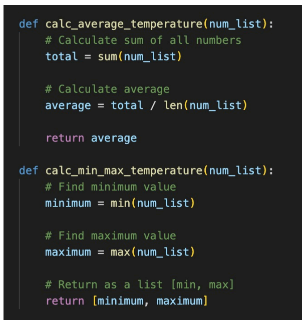
```python
def display_main_menu():
    print("Enter some numbers separated by commas (e.g. 5, 67, 32)")

def get_user_input():
    # Read user input from console
    user_input = input()

    # Split the string by commas into a list of strings
    string_list = user_input.split(",")

    # Convert list of strings to list of floats
    float_list = []
    for num_str in string_list:
        float_list.append(float(num_str))

    return float_list
```

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.



```python
def calc_average_temperature(num_list):
    # Calculate sum of all numbers
    total = sum(num_list)

    # Calculate average
    average = total / len(num_list)

    return average

def calc_min_max_temperature(num_list):
    # Find minimum value
    minimum = min(num_list)

    # Find maximum value
    maximum = max(num_list)

    # Return as a list [min, max]
    return [minimum, maximum]
```

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.

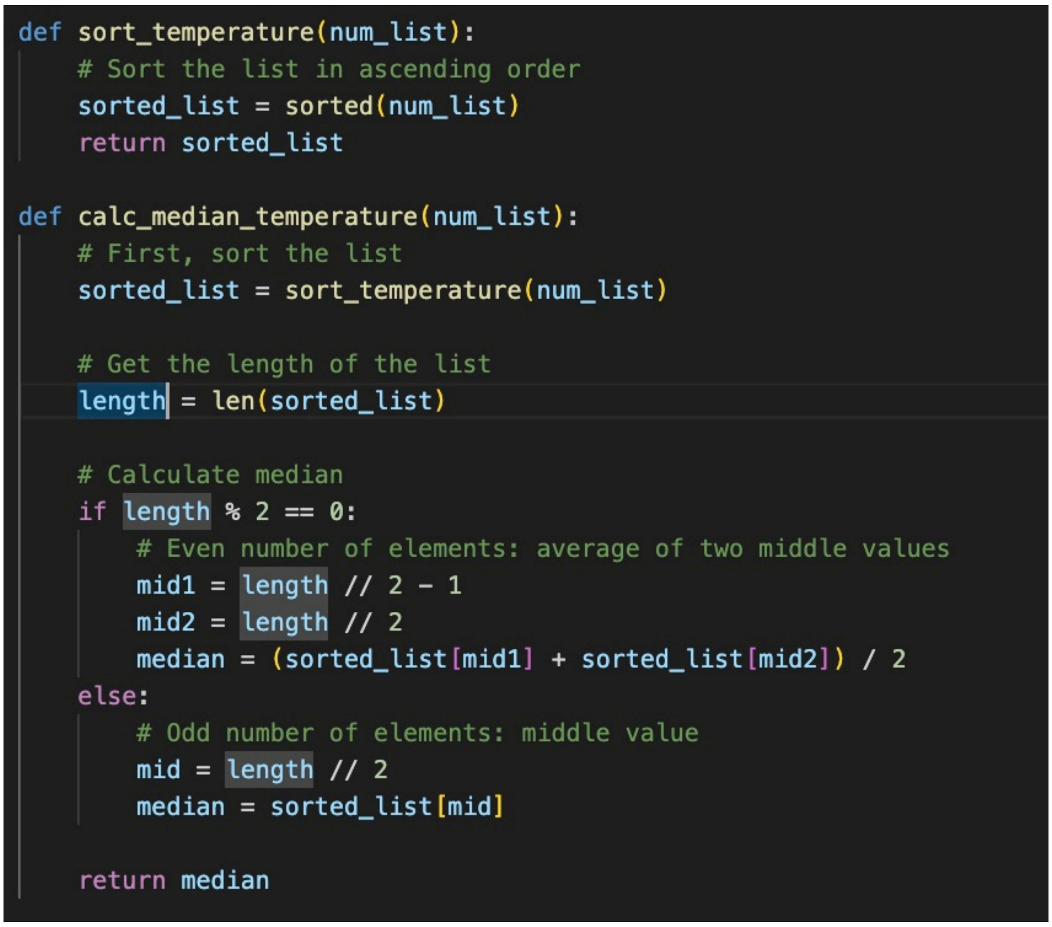
7.8. Create a new **Github release "Lab2_v1.1"** based on the corresponding Git tag.

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.



```python
def sort_temperature(num_list):
    # Sort the list in ascending order
    sorted_list = sorted(num_list)
    return sorted_list

def calc_median_temperature(num_list):
    # First, sort the list
    sorted_list = sort_temperature(num_list)

    # Get the length of the list
    length = len(sorted_list)

    # Calculate median
    if length % 2 == 0:
        # Even number of elements: average of two middle values
        mid1 = length // 2 - 1
        mid2 = length // 2
        median = (sorted_list[mid1] + sorted_list[mid2]) / 2
    else:
        # Odd number of elements: middle value
        mid = length // 2
        median = sorted_list[mid]

    return median
```