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

LABORATORY 3: DEVELOPING SOFTWARE UNIT TESTS

Objectives

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

- Explain the use of PyTest for Software Unit Testing
- Install PyTest in Visual Studio Code
- Define Unit Tests in PyTest

Activities

- Installation and Setup of PyTest
- Configure PyTest for Unit Testing in Visual Studio Code
- Create Software Unit Tests using PyTest

Review

- PyTest is successfully installed in Visual Studio Code.
- PyTest Unit Test cases are created for the Python code for Lab 2 and Lab 3.

Equipment:

Windows OS laptop

Procedures:

1 Create and Execute Software Unit Tests using PyTest

In the previous lab we learned the basics of Python programming, we will now extend this to also introduce the basics of Software Unit Testing based on the Python PyTest Unit Testing framework.

The PyTest Unit Test framework uses standard Python code to define **Unit Test cases** which we can use to test each Python function we have implemented.

Using PyTest, we use the keyword "**assert(…)**" to evaluate the return value of a Python function with the expected value/s returned by the function under test.

Follow the steps below to clone an example of a sample Python script that sorts some numbers in ascending and descending order,

1.1. Go to the C:\ directory of your laptop. Go to "Local_Git_Repository" folder that you have created in lab 1.

1.2. Open a CMD prompt window, and change directory to c:\Local_Git_Repository.

1.3. Clone the Lab 3 Git repository from the link https://github.com/ET0735-DevOps-AIoT/Lab3.git, using the git command below.

```
git clone https://github.com/ET0735-DevOps-AIoT/Lab3.git
```

```
D:\ET0735>git clone https://github.com/ET0735-DevOps-AIoT/Lab3.git
Cloning into 'Lab3'...
remote: Enumerating objects: 37, done.
Receiving objects: 100% (37/37), 5.32
Resolving deltas: 100% (6/6), done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 37 (delta 6), reused 34 (delta 5), pack-reused 0
```

Figure 1 – Clone a repository from GitHub.

1.4. A new folder **"Lab3"** will be created in the C:\Local_Git_Repository directory. This folder is a clone of the remote GitHub repository that you had specified in Step 3.3. You should find the following 7 items in the Lab3 folder:

- .git
- Employee_info.py
- Lab3.py
- price_info.py
- README.md
- Test_Lab3.py

1.5. To open the newly cloned project for experiments, in Visual Studio Code, click "File → Open Folder". When the "Open Folder" window pops up, navigate to C:\Local_Git_Repository, select the Lab3 folder and click "Select Folder".

1.6. The project Lab3 will then be opened in VSCODE IDE.

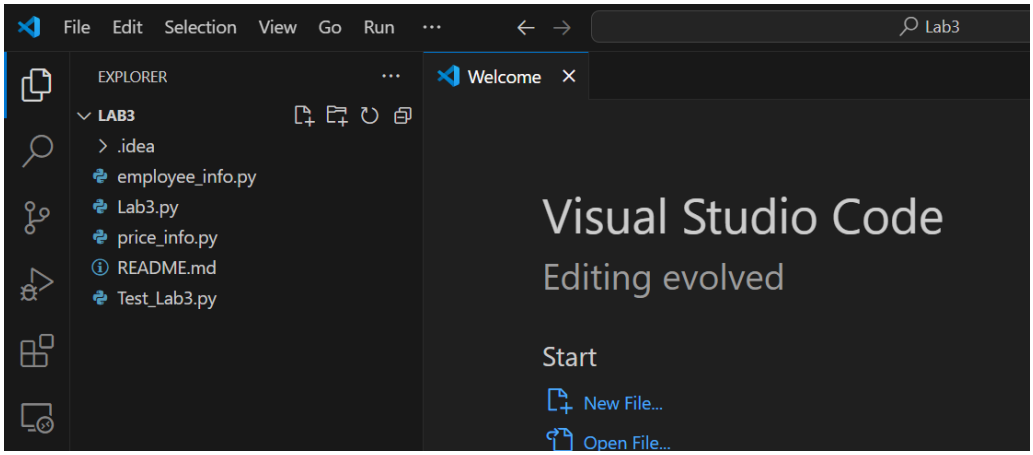

Figure 2 – Project "Lab3" is opened in Visual Studio Code.

2 Configure PyTest for Unit Testing in Visual Studio Code

2.1 Go to the "Testing" tab on the left navigation bar (Figure 3). If the testing tab does not appear on the navigation bar, you need to open the "Test_Lab3.py" file first. We will go into more detail about this file later. After opening the testing window, you will see that there are no tests found in this workspace. This is because PyTest has not been configured yet.

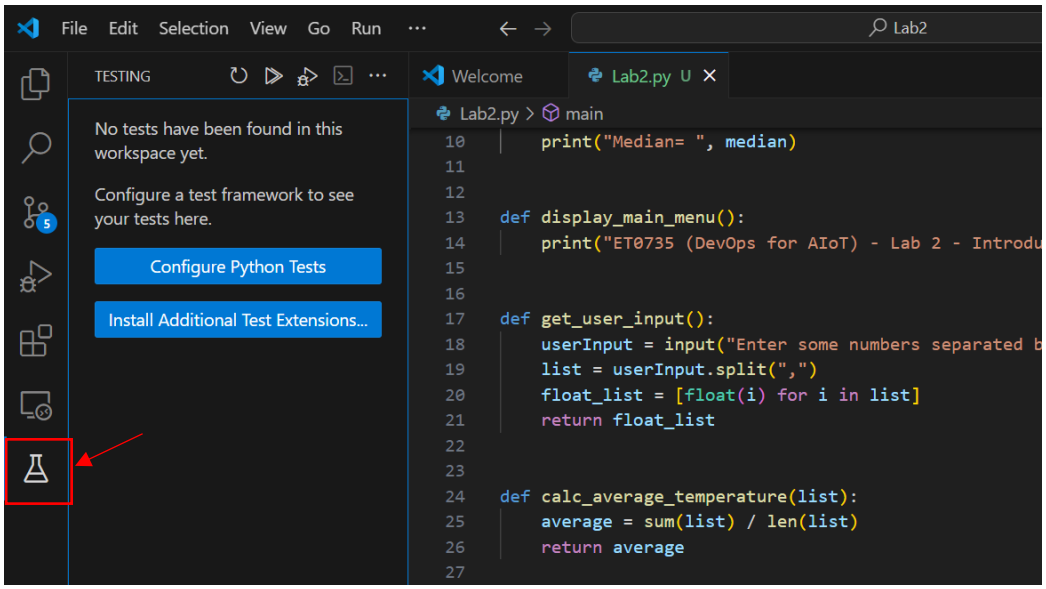

Figure 3 – View the Testing window in Visual Studio Code.

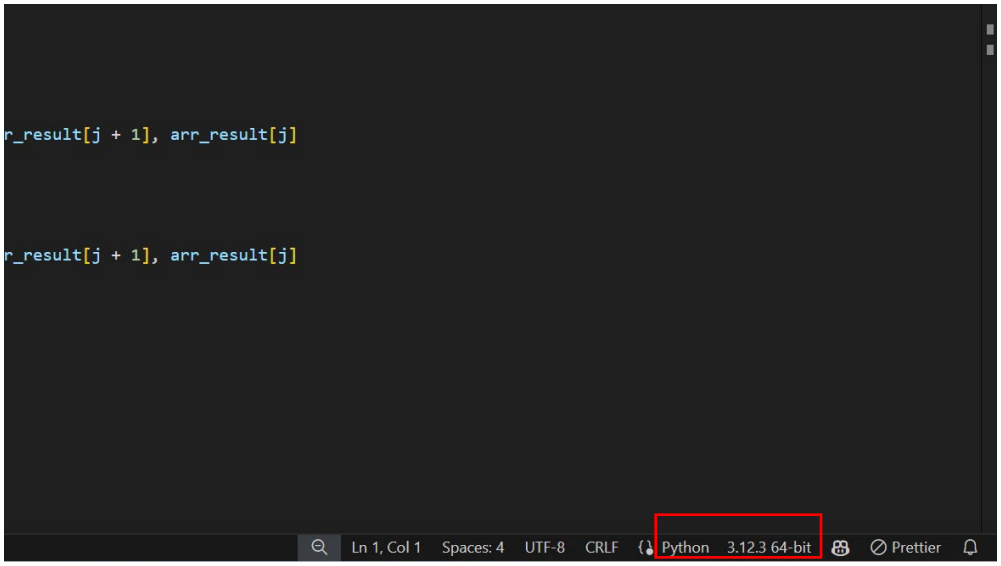

Figure 4 : Python version in VScode

2.2 To configure PyTest, click on “Configure Python Tests”

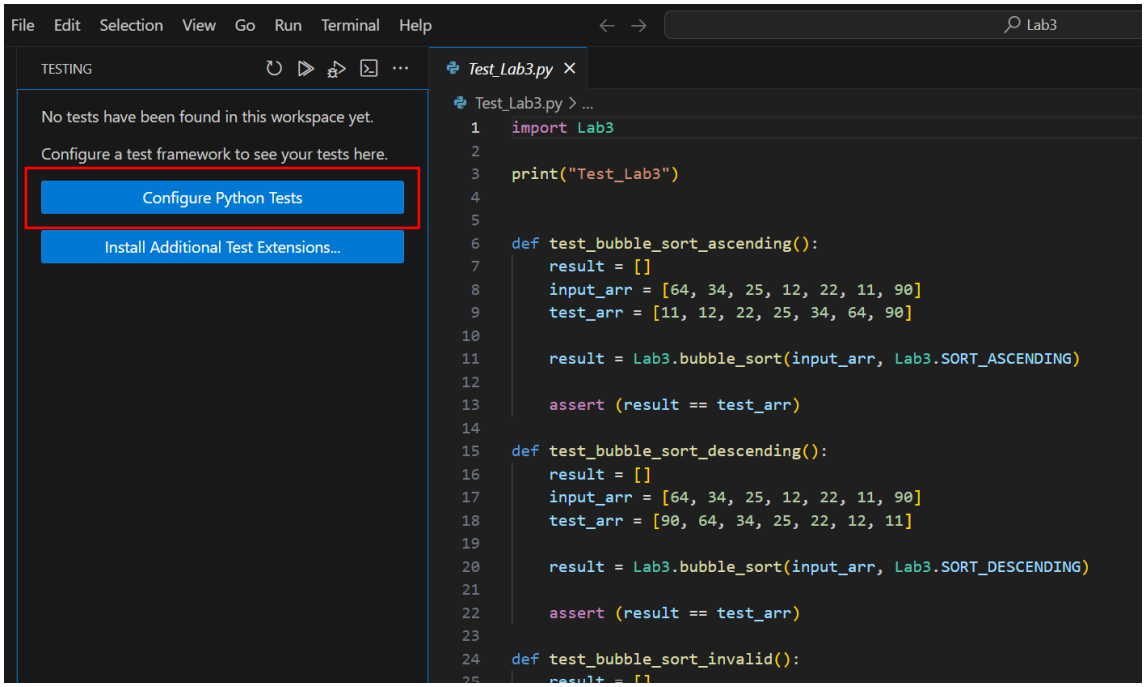

Figure 4 – Configure Python Tests in Visual Studio Code.

2.3 Select pytest in the option and click on pytest and select .root

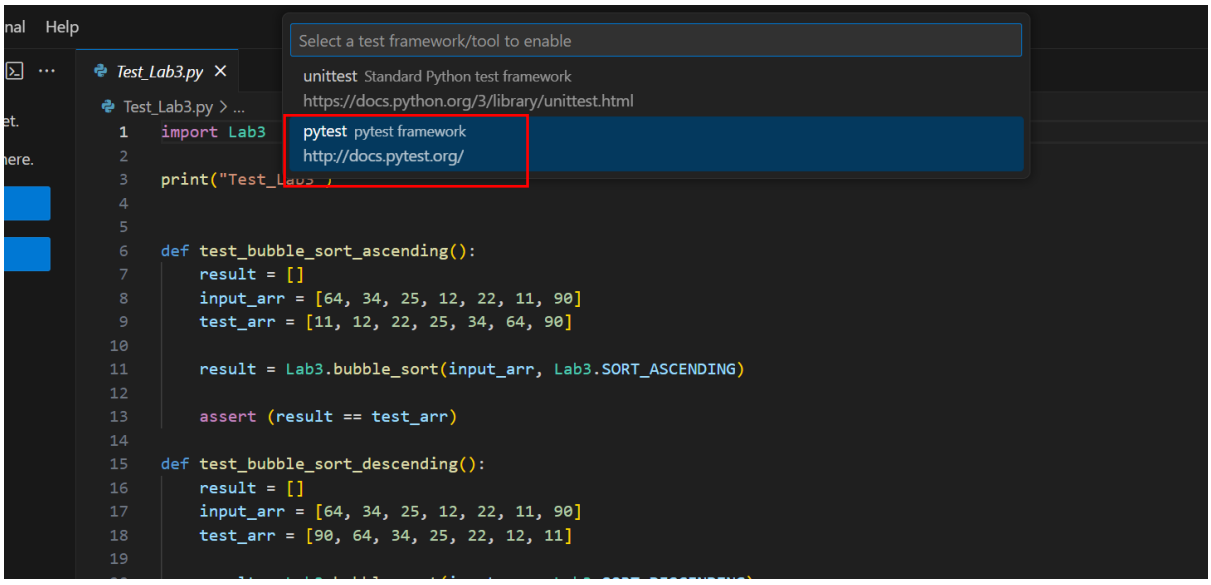

Figure 5 – Select pytest framework

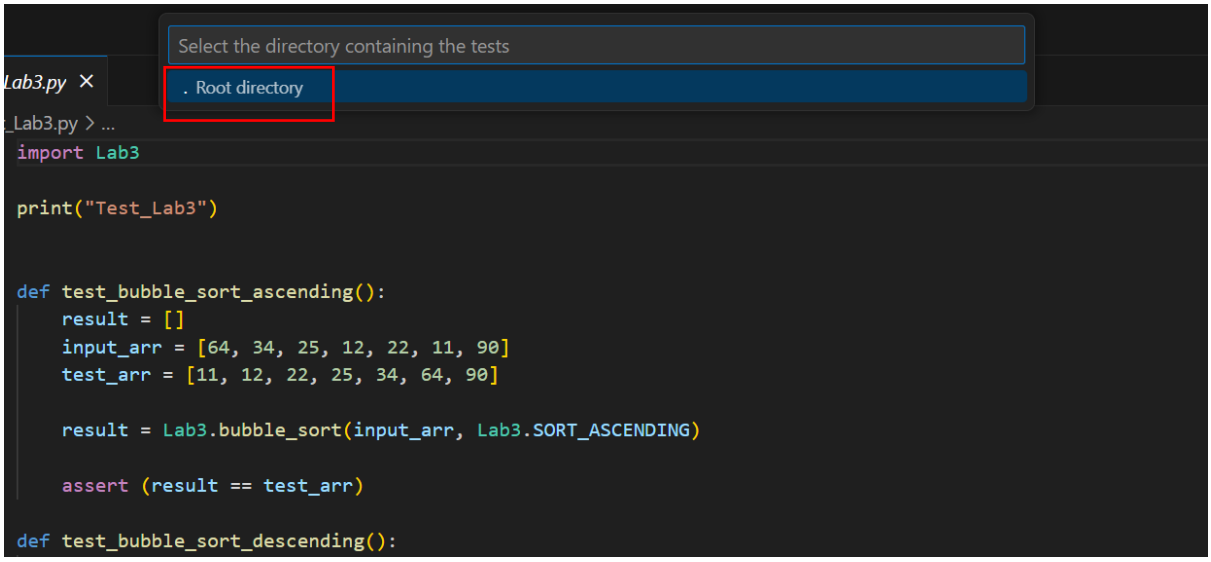

Figure 6 – select .Root directory

2.4 You should notice in the Terminal window (VSCode) pytest is being installed in python using pip.

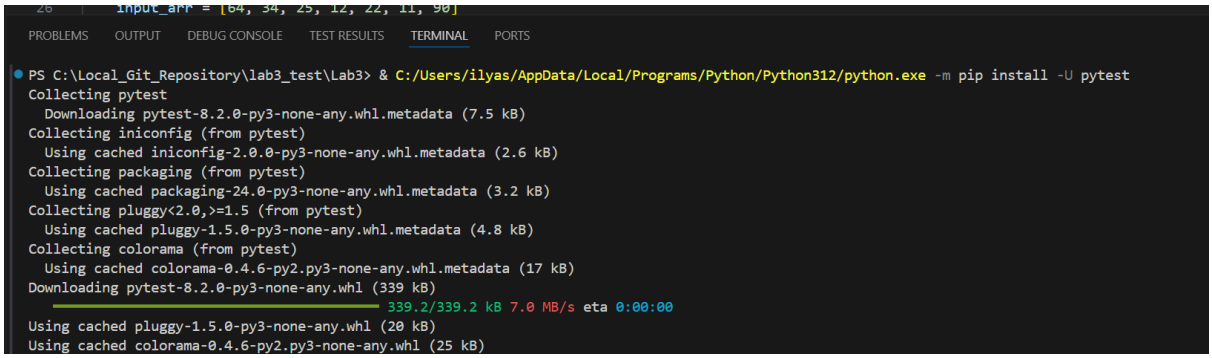

Figure 7 – Opening Settings

2.5 To confirm what modules have been installed in pip, go to your command line console (cmd) and type “pip list” to list out the modules installed in pip (Figure 8). Notice that pytest and additional modules have been installed in your python compiler.

```
C:\Local_Git_Repository\lab3_test\Lab3>pip list
Package    Version
---------- -------
colorama   0.4.6
iniconfig  2.0.0
packaging  24.0
pip        24.0
pluggy     1.5.0
pytest     8.2.0

C:\Local_Git_Repository\lab3_test\Lab3>
```

Figure 8 – Opening Settings

2.6 To configure PyTest, click on “Settings” at the bottom left of the navigation bar. A popup will appear, click on Settings. (Figure 9)

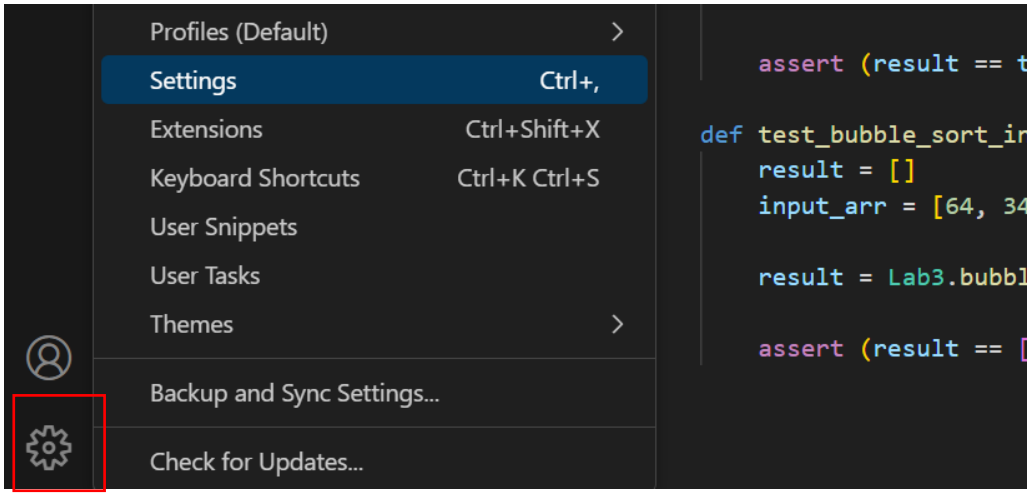

Figure 9 – Opening Settings

2.7 In the search bar, search for “pytest”. This will list all the settings related to pytest. Then, tick the box for “Python > Testing: Pytest Enabled” (Figure 10). This now configures Visual Studio Code to use PyTest as the default Unit Testing Tool.

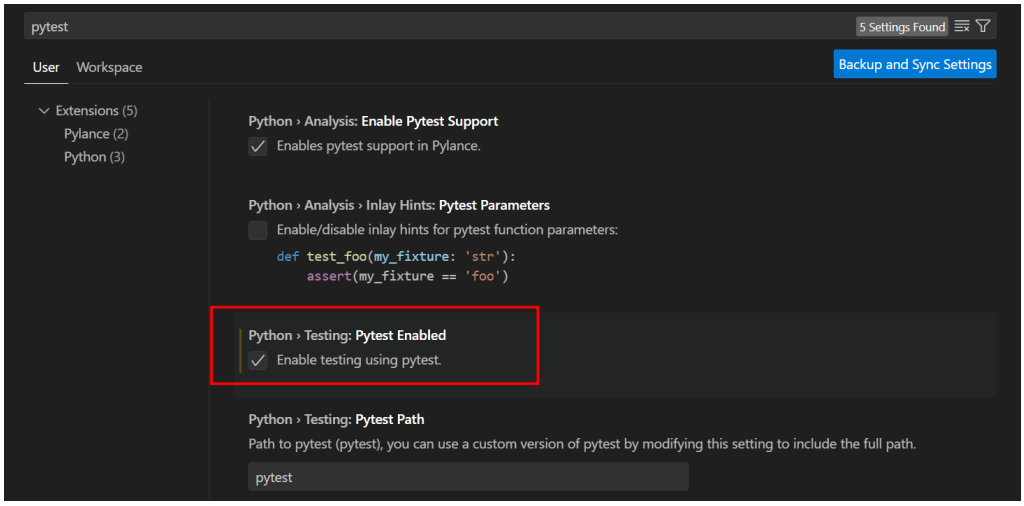

Figure 10 – Enabling PyTest

2.8 After the settings are done, the Lab3 folder should be displayed back in the Testing tab. (Figure 11) You may expand the folders to view all the test functions. If the test functions are not displayed, close lab3 folder in vscode and then open the folder again. Navigate back to Test_lab3.py and select testing to see the test functions.

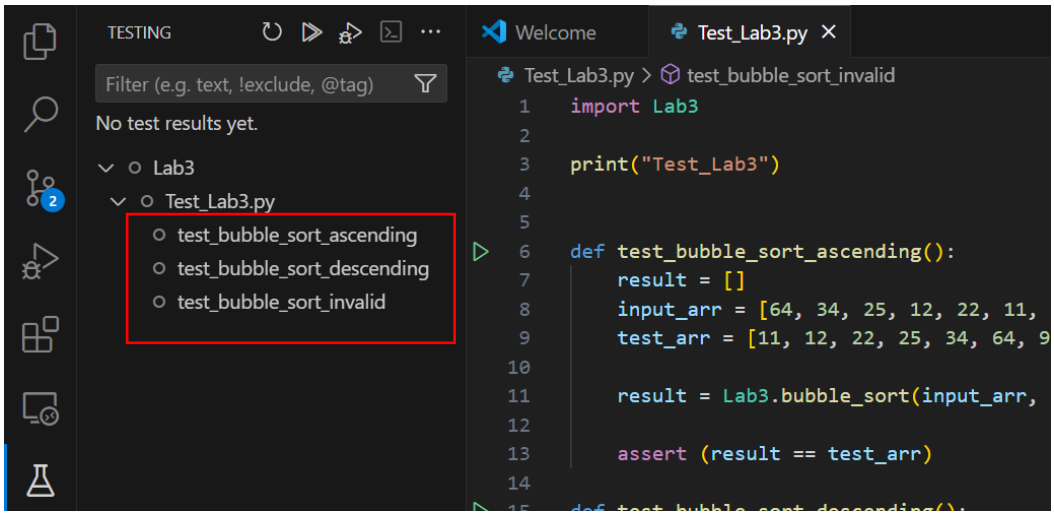

Figure 11 – Pytest in Testing Tab

3 Running Unit Testing using PyTest

3.1. Right-click “Lab3.py” and select “Run Lab3” from the dropdown list. This runs the Python file Lab3.py. Check that the correct console output is displayed to sort a list of numbers in ascending or descending order, as shown below.

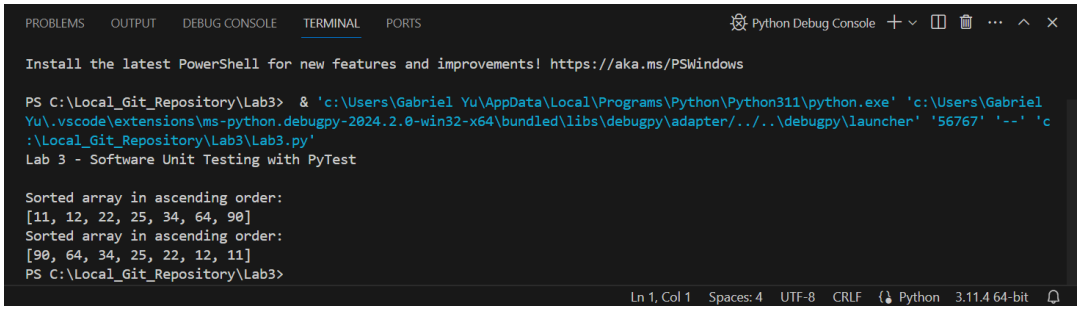

Figure 12 – Console output when running the Lab3.py file.

3.2. In the Lab3 folder, let us head back to the file “Test_Lab3.py”. Double-click it to open.

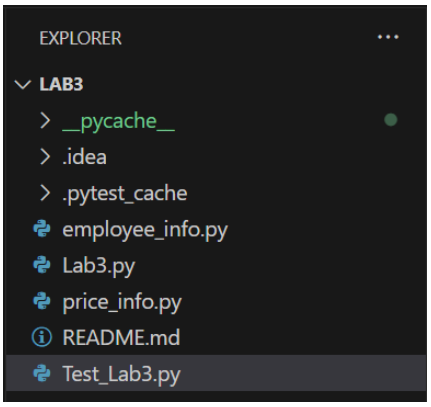

Figure 13 – “Test_Lab3.py” is a Python file that contains the PyTest unit tests.

“Test_Lab3.py” is a Python file where we have defined the PyTest Unit Test to check that all the functions implemented in “Lab3.py” are correct

In “Test_Lab3.py”, the following PyTest Unit Test cases are defined,

- Test Case 1 → test_bubble_sort_ascending()
- Test Case 2 → test_bubble_sort_descending()
- Test_Case3 → test_bubble_sort_invalid()

Test Cases 1 and 2 are known as “Positive” test cases where the test cases check for valid input combinations versus the expected result.

Test Case 3 is for checking what happens when invalid or unexpected inputs are passed into the function “bubble_sort()”.

3.3. For each PyTest test, notice the format and syntax below where each test cases ends with an “assert” statement.

```
def test_bubble_sort_ascending():
    result = []
    input_arr = [64, 34, 25, 12, 22, 11, 90]
    test_arr = [11, 12, 22, 25, 34, 64, 90]
    result = Lab3.bubble_sort(input_arr, Lab3.SORT_ASCENDING)

    assert (result == test_arr)
```

The assert statement in PyTest basically returns a Boolean value True of False of the condition asserted or checked is True or False.

In Software Unit Testing, we basically want to verify and check that the function under test returns some predefined expected values based on a set of corresponding inputs.

3.4. To execute Pytest unit test cases inside the Visual Studio Code IDE, there are 2 main methods,
- Execute all Pytest Unit Test cases within a Python Script
- Execute selection individual test cases

If you want to execute all PyTest test cases, just enter the testing tab from **Step 2.1** and click on “Run Test” (Figure 14) to run all test functions within the selected Python file.

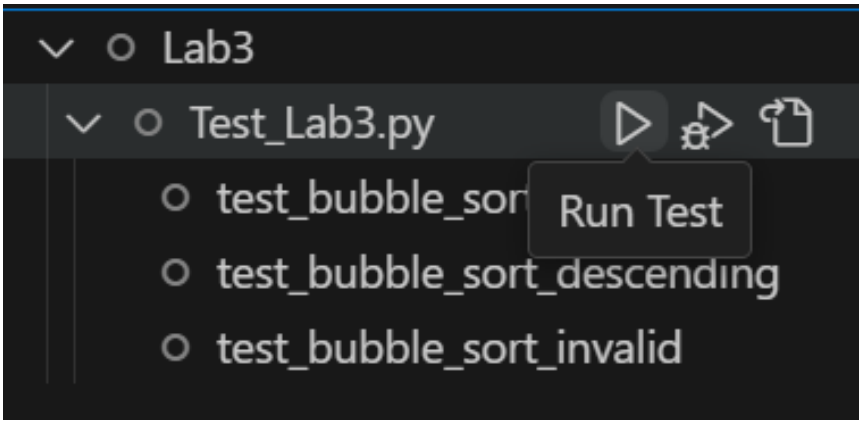

Figure 14 – To execute all PyTest test cases.

3.5. To run selected PyTest functions individually in Visual Studio Code, just click on the green icon on the left margin of the PyTest function you want to execute (Figure 15).

```
5
6   def test_bubble_sort_ascending():
7       result = []
8       input_arr = [64, 34, 25, 12, 22, 11, 90]
9       test_arr = [11, 12, 22, 25, 34, 64, 90]
10
11      result = Lab3.bubble_sort(input_arr, Lab3.SORT_ASCENDING)
12
13      assert (result == test_arr)
14
15  def test_bubble_sort_descending():
16      result = []
17      input_arr = [64, 34, 25, 12, 22, 11, 90]
18      test_arr = [90, 64, 34, 25, 22, 12, 11]
19
20      result = Lab3.bubble_sort(input_arr, Lab3.SORT_DESCENDING)
21
22      assert (result == test_arr)
23
24  def test_bubble_sort_invalid():
25      result = []
26      input_arr = [64, 34, 25, 12, 22, 11, 90]
27
28      result = Lab3.bubble_sort(input_arr, 3)
29
```

Figure 15 – To run selected PyTest functions individually in Visual Studio Code.

3.6. Try to execute all PyTest test cases in the Test_Lab3.py file, using the approach shown in Figure 14. After running all the PyTest Unit Test case, a Test Status report will be shown in Visual Studio Code, summarizing the test cases that were executed and the test results.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  TEST RESULTS  PORTS

cending', 'c:\\Local_Git_Repository\\Lab3\\Test_Lab3.py::test_bubble_sort_descen
ding', 'c:\\Local_Git_Repository\\Lab3\\Test_Lab3.py::test_bubble_sort_invalid']
============================= test session starts =============================
platform win32 -- Python 3.11.4, pytest-7.4.3, pluggy-1.3.0
rootdir: c:\Local_Git_Repository\Lab3
collected 3 items

Test_Lab3.py ...                                                        [100%]

============================== 3 passed in 0.02s ==============================
Finished running tests!

Test run at 3/14/2024, 11:41:53 AM
test_bubble_sort_ascending
test_bubble_sort_descending
test_bubble_sort_invalid

Ln 30, Col 26  Spaces: 4  UTF-8  CRLF  Python  3.11.4 64-bit
```

Figure 16 – Test Status report, summarizing the test cases executed and the test results

Exercise 1

Based on the BMI Python code in "bmi.py" that you implemented in the previous Lab 2, we will now implement suitable PyTest cases to verify that your function "calculate_bmi()" is implemented correctly based on the requirements in Table 2.

(a) Open your Lab 2 Visual Studio Code project and access Python file "bmi.py", modify the function "calculate_bmi()" to also return the following values defined in Table 1 below,

Weight Classification	Return Value
Under weight	-1
Normal weight	0
Over weight	1

Table 1

(b) After successfully implementing the above, commit your changes and push to your lab2 github repository.

(c) In your lab3 Visual Studio Code project, using the Lab 3 Git repository, create a new Git submodule based on your Lab 2 Github repository which you have updated in the task above. Write down the Git command/s used in the box below,

(d) Using the existing Lab3 Visual Studio Code project, create a new PyTest file "Test_bmi.py" which we will use to define some PyTest Unit Test cases to verify the Lab 2 BMI Python application code.

(e) Add a suitable "import" statement at the top of the new "Test_bmi.py" file to import your Lab 2 "bmi.py" file based on your own folder structure for Lab 2. For example you can add the following import if you had stored all your Lab 2 files in a folder called "ET0735_Lab2".

```
import ET0735_Lab2.bmi as bmi
```

(f) In "Test_bmi.py", create and implement the PyTest Unit Test functions below based on the requirements defined in Table 2

test_bmi_normal_weight()
test_bmi_over_weight()
test_bmi_under_weight()

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

Table 2

(g) Once the unit test functions are implemented and tested to work, you will need to commit your changes to your local repository to track your progress.
Write the command in the box below to **check** the files that you need to add and commit.

Did you notice in the "untracked files" section there is a new folder __pycache__?

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        __pycache__/

no changes added to commit (use "git add" and/or "git commit -a")
```

Question : What files are found in this folder? Do you need to include these files and folder in your git repository? State your reasons.

(h) In order for git to ignore certain specific files or folders, you will need to add a new file .gitignore.
In your code folder create a new file .gitignore. To ignore __pycache__ folder, you can add the following code in .gitignore file:

```
__pycache__/
```

(i) Add and commit the files that you have modified for this section with a message stating your implementation. Do not push your changes to remote repository yet.

Exercise 2

Based on the requirements defined in Table 3 below, analyse the Python code implemented in Lab3.py and the PyTest Unit Test cases defined in Test_Lab3.py

If you find any parts of the Python code implementation that does not match the requirements defined in Table 3, then please proceed to modify the Python code to match the requirements defined in Table 3.

Requirement ID	Requirement	PyTest Function/s
REQ-01	If < 10 numbers are entered and "SORT_ASCENDING" is passed to the function "bubble_sort()", then the function returns the list of numbers sorted in ascending order.	
REQ-02	If < 10 numbers are entered and "SORT_DESCENDING" is passed to the function "bubble_sort()", then the function returns the list of numbers sorted in descending order.	
REQ-03	If >= 10 numbers are entered, the function "bubble_sort()" shall **return the integer value 1**	
REQ-04	If 0 numbers are entered, the function "bubble_sort()" shall **return the integer value 0**	
REQ-05	If any of the values entered on the command line console are **not integers**, the function "bubble_sort()" shall **return the integer value 2**	

Table 3

(a) Using Table 3, update the column "PyTest Function/s" for requirements that have an existing Unit Test case defined in Test_Lab3.py.

(b) For all requirements that do not have any test case defined in Table 3, implement the missing Unit Test Cases in Test_Lab3.py.

(c) If any of the requirements defined in Table 3 are either not implemented or implemented differently in the existing code, then you need to update the Python code to match the requirements in Table 3**.**

(d) Execute all PyTest Unit Test cases and check that all requirements defined in Table 3 are fully tested and pass the PyTest Unit Test Cases.

Add, Commit and Push changes to a new Github repository

Since we cloned the initial Lab 3 Github repository, the remote URL has been already been configured to "`https://github.com/ET0735-DevOps-AIoT/Lab3.git`" and needs to be changed to the URL of new Lab 3 Github repository

(e) Create a new repository "Lab3" in Github

(f) View the current remote URL with the following Git command and write the results of the git command below,

```
git remote –v
```

(g) Update the remote origin to the URL of your new Github repository using the command below replacing {URL} with the URL to your new Github "Lab3" repository

```
git remote set-url origin {URL}
```

(h) Add and Commit all changes to the local repository

(i) Create a new Git Tag "Lab3_v1.0"

(j) Push all changes and the new Tag to Github

(k) For any PyTest Unit Test cases that fail, implement the bug fix changes in Lab3.py and the run the PyTest Unit Test cases again until all pass.

(l) Add and Commit the updated code implementation in Lab3.py

(m) Create a new Git Tag "Lab3_v2.0" and Push all changes to Github

Exercise 3

Based on the exercises in Lab2.py, create PyTest Unit Test cases for the following functions,

- find_min_max()
- calc_average()
- calc_median_temperature()

(a) Open the Visual Studio Code project Lab2, create a new file "Test_Lab2.py"

(b) Create the new test cases for the functions above in "Test_Lab2.py"

(c) Commit and Push all changes to Github for Lab 2 Git Repository

(d) Create a new Software Release "Lab2_v3.0" in Github

Exercise 4

In this exercise, you will be working with python dictionaries. Dictionaries are a versatile and powerful built-in data structure in the Python programming language. Dictionaries are unordered, mutable, and store key-value pairs, also known as associative arrays or hash maps in other programming languages. Each key in a dictionary is unique, and it maps to a corresponding value, allowing for quick and efficient data retrieval and manipulation.

(a) Now open your lab3 Visual Studio Code project and open the file price_info.py. This python file contains 2 dictionaries, one for the price list of fruits named price_list and another for the quantity of fruits that was purchased. The key and values are distinguished by the ":". Any value can be accessed by using the corresponding key.

(b) There are 2 functions created using the dictionaries provided. For the function **cost_of_fruits**, given the key for the fruit and quantity, the total cost of the purchase is printed. For the function **total_cost_shopping**, the 2 dictionaries are traversed and total cost of the purchase is computed based on the quantity in quantity_list for each fruit. Implement the missing code in **total_cost_shopping**. Test the code to make sure it works correctly.

(c) Create a new file test_price_info and write the unit test cases for the following functions:

- total_cost_shopping()
- cost_of_fruit()

(d) commit all changes and push your implemented code to your lab 3 repository.

(e) Create new Software Release "Lab3_v4.0" in Github

Exercise 5

In this exercise, you will be working with list of python dictionaries. A list of dictionaries can be used as a simple form of database, where each dictionary in the list represents a record or row in the database, and the keys in each dictionary represent the fields or columns of the database.

For example, you might define a list of dictionaries to represent a table of customer information, where each dictionary in the list represents a customer record with fields such as name, email, phone, and address.

Here is an example of customer records:

```
customers = [

    {'name': 'Alice', 'email': 'alice@example.com', 'phone': '555-1234', 'address': '123 Main St'},

    {'name': 'Bob', 'email': 'bob@example.com', 'phone': '555-5678', 'address': '456 Maple Ave'},

    {'name': 'Charlie', 'email': 'charlie@example.com', 'phone': '555-9012', 'address': '789 Oak Rd'},

]
```

Once you have a list of dictionaries like this, you can easily perform various database operations on it, such as adding or removing records, querying records by specific fields or values, and updating fields in specific records.

You will need to implement some functions for an employee information tracking system python application in this exercise.

(a) Now open your lab3 Visual Studio Code project and open the file employee_info.py. The application with some not implemented functions are available. employee_data is a dictionary with key values for “name”, “age”, “department” and “salary”. While the corresponding values for each key is separated by a (‘:’). Some common methods for working with dictionaries include keys(), values(), items(), get(), update(), and pop(). These methods make it easy to interact with and manipulate dictionary data.

(b) In this simple application, there are 4 options available:

- Display all the records of employee
- Display the average salary
- Display the employee details within certain age range
- Display employees within a particular department.

(c) The first requirement is to calculate the average salary of all the employees in the database. The function stub : calculate_average_salary is already provided. Implement the code and test. Refer to get_employees_by_age_range function for usage of dictionaries. After successfully implementing the function. Commit the code in your local repository with an appropriate message for your reference.

(d) The second requirement is to display employees in a given department. The department name is provided as an parameter to the function get_employees_by_dept. Implement and test for this function. Subsequently commit the code in your local repository.

(e) Create a new file test_employee_info and write pytest cases for the following functions:

   a. get_employees_by_age_range
   b. calculate_average_salary
   c. get_employees_by_dept

(f) Commit all changes and push all your changes to lab3 repository.

(g) Create new Software Release “Lab3_v5.0” in Github