

Gabriel Q. Escobido

BSCpE – 2A

SOFTWARE DESIGN

Laboratory Exercise No. 9

Title: Software Testing Techniques: Unit and Integration Testing

Brief Introduction

This exercise introduces unit and integration testing in software development. Students will learn to write and execute test cases using Python's unittest framework.

Objectives

- Understand the purpose and scope of unit and integration testing.
- Write Python test cases to validate individual components and their interactions.
- Analyze the importance of automated testing in software development.

Detailed Discussion

- **Unit Testing:** Verifies the correctness of individual components (e.g., functions, classes).
- **Integration Testing:** Ensures that combined components function as expected.
- **Test Automation:** Automates repetitive test execution to save time and reduce errors.

Testing Type	Description	Example
Unit Testing	Validates individual functions or components.	Testing a single function
Integration Testing	Tests interaction between multiple modules.	Testing database and API

Materials

- Python environment
- unittest module

Procedure

1. Write unit tests for a simple arithmetic module:

```
# arithmetic.py
```

```
def add(a, b):
```

```
    return a + b
```

```
def multiply(a, b):
```

```
    return a * b
```

```
# test_arithmetic.py
```

```
import unittest
```

```
from arithmetic import add, multiply
```

```
class TestArithmetic(unittest.TestCase):
```

```
    def test_add(self):
```

```
        self.assertEqual(add(2, 3), 5)
```

```
    def test_multiply(self):
```

```
        self.assertEqual(multiply(2, 3), 6)
```

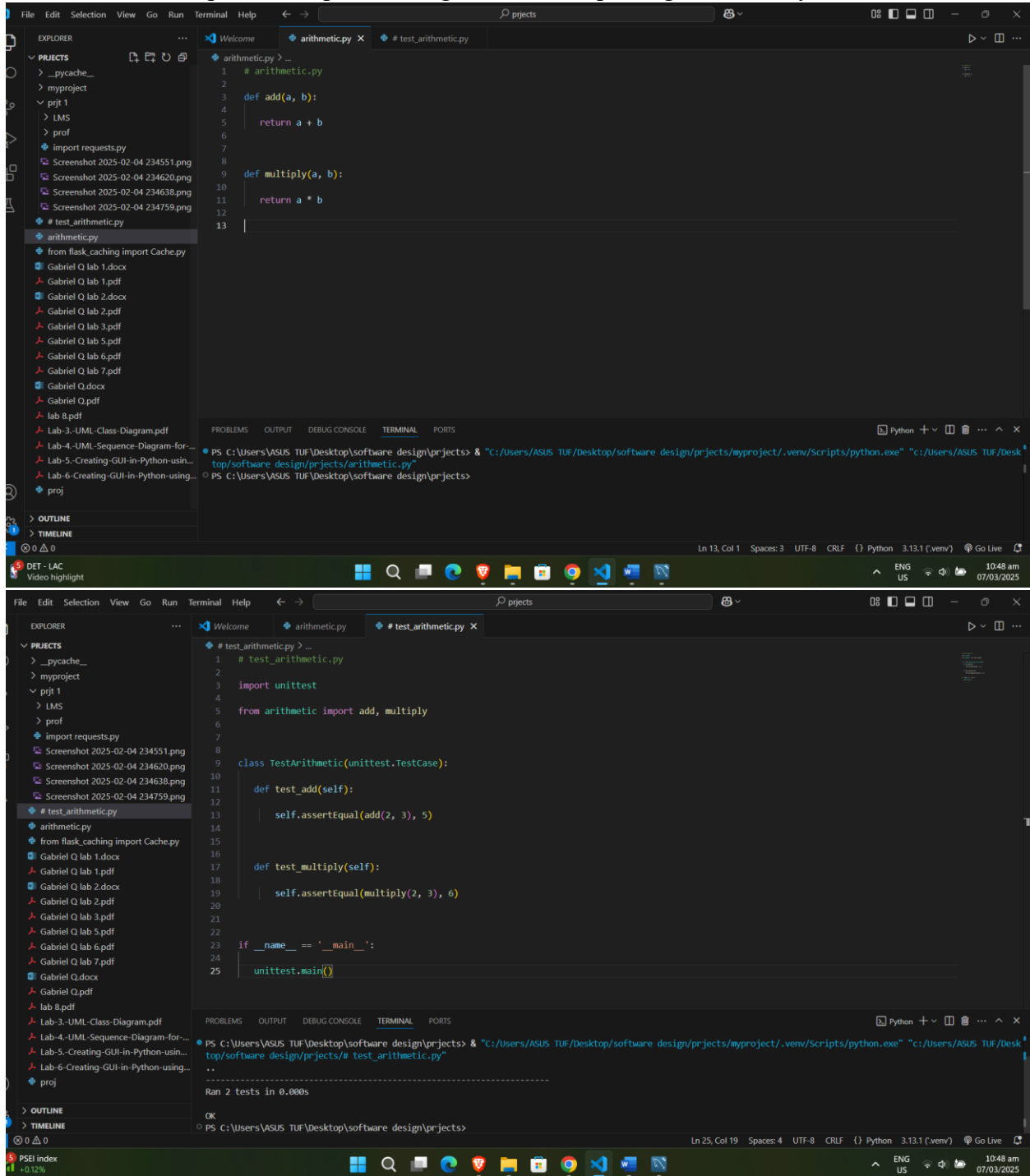
```
if __name__ == '__main__':
```

```
    unittest.main()
```

1. Extend the tests to include integration scenarios (e.g., API + database testing).

Results

Submit the test scripts and output showing all test cases passing successfully.



The image shows two screenshots of a Visual Studio Code editor window. The top screenshot displays the `arithmetic.py` file with the following code:

```
1 # arithmetic.py
2
3 def add(a, b):
4     return a + b
5
6
7
8
9 def multiply(a, b):
10     return a * b
11
12
13
```

The bottom screenshot displays the `test_arithmetic.py` file with the following code:

```
1 # test_arithmetic.py
2
3 import unittest
4
5 from arithmetic import add, multiply
6
7
8
9 class TestArithmetic(unittest.TestCase):
10
11     def test_add(self):
12         self.assertEqual(add(2, 3), 5)
13
14
15
16     def test_multiply(self):
17         self.assertEqual(multiply(2, 3), 6)
18
19
20
21
22
23 if __name__ == '__main__':
24     unittest.main()
25
```

The terminal output at the bottom of the second screenshot shows the command executed and the test results:

```
PS C:\Users\ASUS TUF\Desktop\software design\projects> & "C:/Users/ASUS TUF/Desktop/software design/projects/myproject/.venv/Scripts/python.exe" "C:/Users/ASUS TUF/Desktop/software design/projects/# test_arithmetic.py"
...
Ran 2 tests in 0.000s
OK
PS C:\Users\ASUS TUF\Desktop\software design\projects>
```

Follow-Up Questions

1. What is the difference between unit and integration testing?

ANSWER: Unit testing focuses on testing individual components or functions in isolation, ensuring that each part works correctly on its own. Integration testing, on the other hand, focuses on testing how multiple components or systems work together, ensuring that they interact correctly when integrated.

2. Why is automated testing important in large-scale software projects?

ANSWER: Automated testing ensures that code changes do not introduce new bugs, speeds up the testing process, and provides consistency in running tests. In large-scale projects, automated tests help save time and effort, improve code quality, and allow for frequent code changes without compromising stability.

3. Can you identify scenarios where integration testing is critical?

ANSWER: Integration testing is critical when multiple systems or components interact with each other, such as when a web service communicates with a database or an API interacts with external third-party services. It ensures that these systems work together as expected and helps catch issues that may not be apparent when testing components in isolation.