

Gabriel Q. Escobido

BSCoE – 2A

SOFTWARE DESIGN

Laboratory Exercise No. 3 CH2

Title: Exploring Programming Paradigms

Brief Introduction

Programming paradigms define the style and structure of writing software programs. This exercise introduces imperative, object-oriented, functional, declarative, event-driven, and concurrent programming paradigms and their applications.

Objectives

- Compare and contrast different programming paradigms.
- Implement examples using Python for multiple paradigms.
- Explore practical use cases for each paradigm.

Detailed Discussion

Programming Paradigm	Description	Example Applications
Imperative	Focuses on step-by-step instructions.	Low-level programming tasks
Object-Oriented	Organizes code using objects and classes.	GUI applications, simulations
Functional	Emphasizes mathematical functions and immutability.	Data analysis, AI
Declarative	Specifies what to do without describing how to do it.	SQL, configuration files
Event-Driven	Responds to events like clicks, signals, or messages.	GUIs, games
Concurrent	Manages multiple computations at the same time.	Web servers, parallel processing

Materials

- Python environment
- VS Code IDE

Procedure

1. Implement imperative programming in Python:

Imperative programming example

```
nums = [1, 2, 3, 4, 5]
```

```
total = 0
```

```
for num in nums:
```

```
    total += num
```

```
print("Total:", total)
```

1. Create a simple object-oriented program:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        return f'Hello, my name is {self.name} and I am {self.age} years old.'

p = Person("Alice", 25)
print(p.greet())
```

1. Write a functional programming example using Python's map and filter:

```
nums = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, nums))
even_nums = list(filter(lambda x: x % 2 == 0, nums))
print("Squared Numbers:", squared)
print("Even Numbers:", even_nums)
```

1. Showcase event-driven programming using Tkinter:

```
import tkinter as tk

def on_button_click():
    label.config(text="Button clicked!")

root = tk.Tk()
```

```
button = tk.Button(root, text="Click me!", command=on_button_click)
```

```
button.pack()
```

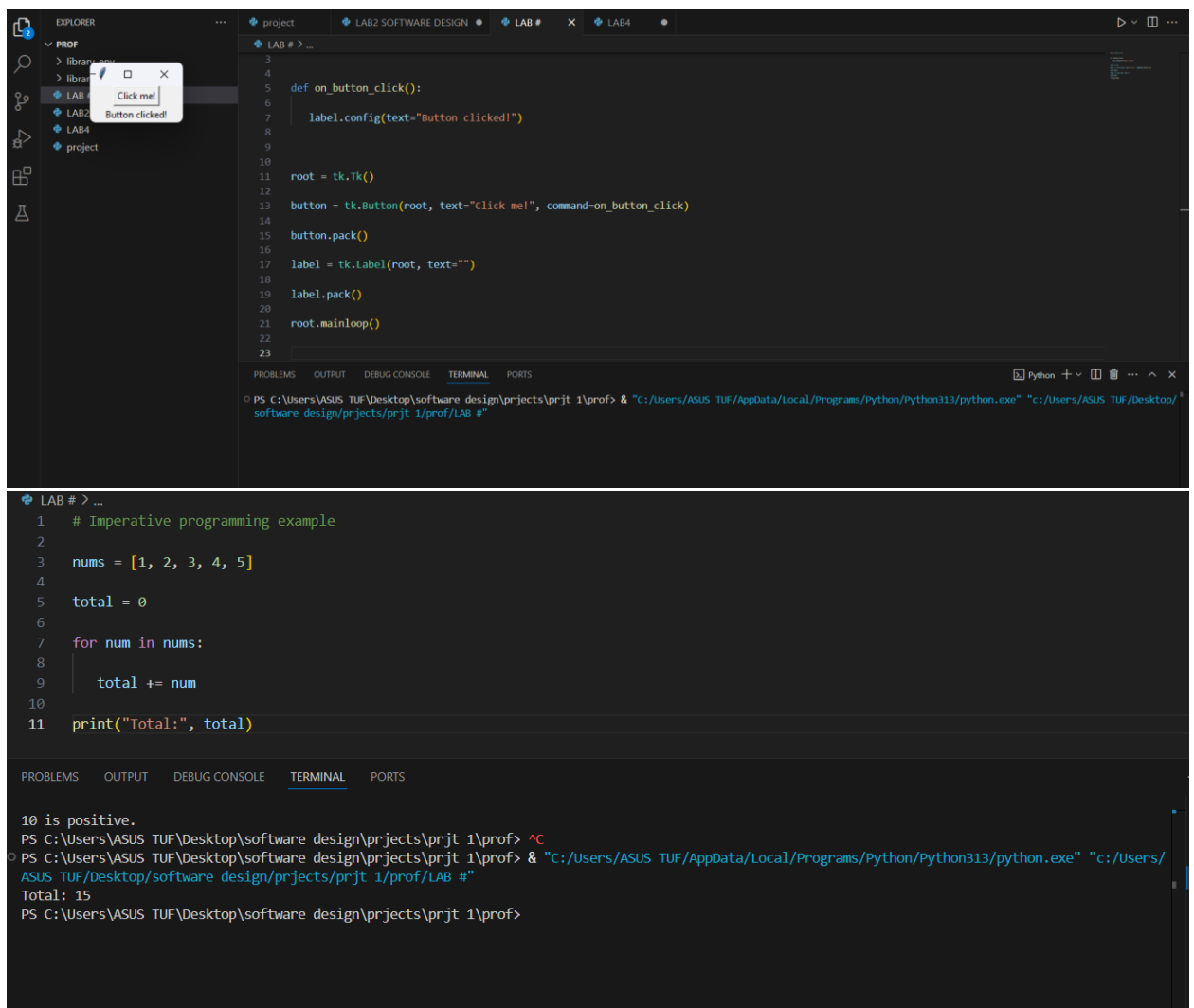
```
label = tk.Label(root, text="")
```

```
label.pack()
```

```
root.mainloop()
```

1. Discuss concurrency with the threading module.

RESULT



The image shows a screenshot of a VS Code editor with two windows. The top window displays a Tkinter GUI with a button labeled "Click me!" and a label. The bottom window shows a Python script for an imperative programming example.

Top Window (GUI):

```
3
4
5 def on_button_click():
6     label.config(text="Button clicked!")
7
8
9
10
11 root = tk.Tk()
12
13 button = tk.Button(root, text="Click me!", command=on_button_click)
14
15 button.pack()
16
17 label = tk.Label(root, text="")
18
19 label.pack()
20
21 root.mainloop()
22
23
```

Bottom Window (Python Script):

```
1 # Imperative programming example
2
3 nums = [1, 2, 3, 4, 5]
4
5 total = 0
6
7 for num in nums:
8     total += num
9
10
11 print("Total:", total)
```

Terminal Output:

```
10 is positive.
PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof> ^C
PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof> & "C:/Users/ASUS TUF/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/ASUS TUF/Desktop/software design/projects/prjt 1/prof/LAB #"
Total: 15
PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof>
```

The image contains two screenshots of a Python IDE, likely VS Code, showing code execution. The top screenshot displays a Python class named 'Person' with an 'init' method and a 'greet' method. The 'init' method sets 'self.name' and 'self.age'. The 'greet' method returns a string. An instance 'p' is created with name 'Alice' and age 25, and 'p.greet()' is called. The terminal output shows 'Hello, my name is Alice and I am 25 years old.' The bottom screenshot shows a list of numbers [1, 2, 3, 4, 5] being processed with 'map' and 'filter' functions. 'map' squares the numbers, and 'filter' keeps even numbers. The terminal output shows 'Squared Numbers: [1, 4, 9, 16, 25]' and 'Even Numbers: [2, 4]'.

```
LAB # > ...
1 class Person:
2
3     def __init__(self, name, age):
4
5         self.name = name
6
7         self.age = age
8
9
10
11     def greet(self):
12
13         return f"Hello, my name is {self.name} and I am {self.age} years old."
14
15
16
17 p = Person("Alice", 25)
18
19 print(p.greet())
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [] ... ^ x

PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof> & "C:\Users\ASUS TUF\AppData\Local\Programs\Python\Python313\python.exe" "c:\Users\ASUS TUF/Desktop/software design/projects/prjt 1/prof/LAB #"
Hello, my name is Alice and I am 25 years old.
PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof>

```
LAB # > ...
1 nums = [1, 2, 3, 4, 5]
2
3 squared = list(map(lambda x: x**2, nums))
4
5 even_nums = list(filter(lambda x: x % 2 == 0, nums))
6
7 print("Squared Numbers:", squared)
8
9 print("Even Numbers:", even_nums)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [] ... ^ x

● PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof> & "C:\Users\ASUS TUF\AppData\Local\Programs\Python\Python313\python.exe" "c:\Users\ASUS TUF/Desktop/software design/projects/prjt 1/prof/LAB #"
Squared Numbers: [1, 4, 9, 16, 25]
Even Numbers: [2, 4]
○ PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof>

Follow-Up Questions

1. What are the key differences between imperative and declarative programming?

ANS: Imperative programming focuses on *how* to perform tasks with explicit instructions and control flow, while declarative programming

focuses on *what* the desired outcome should be, leaving the system to determine how to achieve it.

2. In which scenarios would you prefer functional programming?

ANS: In which scenarios would you prefer functional programming?

3. How can concurrency improve software performance?

ANS: Concurrency can improve software performance by allowing multiple tasks to run simultaneously, making better use of system resources, reducing wait times, and improving overall throughput, especially in multi-core or distributed environments.