Segales, Ace Niño B.

BSCPE 2A

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))
```

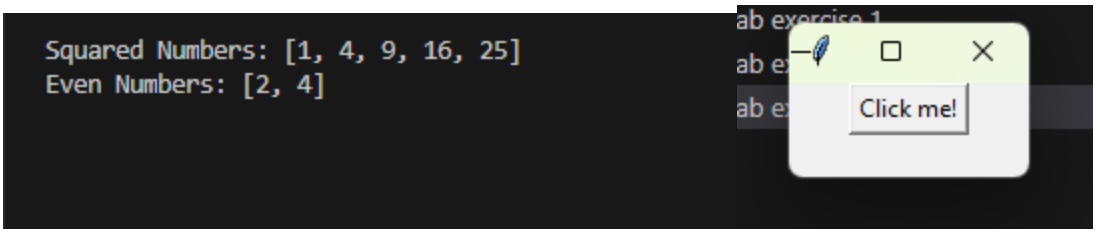
```
even_nums = list(filter(lambda x: x % 2 == 0, nums))
print("Squared Numbers:", squared)
print("Even Numbers:", even_nums)
```

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()
```

lab exersice 3 > ...

```python
# Imperative programming example
nums = [1, 2, 3, 4, 5]

total = 0
for num in nums:
    total += num

print("Total:", total)

# Class definition
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

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

# Functional programming example
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)

# GUI example
import tkinter as tk

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

if __name__ == "__main__":  # Main guard
    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.

Follow-Up Questions

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

The key differences between imperative and declarative programming lie in their focus, with imperative programming emphasizing how to perform tasks through explicit control flow and state management, while declarative programming emphasizes what the desired outcome is, abstracting away the details of execution, leading to more concise and readable code.

2. In which scenarios would you prefer functional programming?

Because it emphasizes pure functions, immutability, and higher-order functions that improve code clarity, maintainability, and reliability, functional programming is preferred in scenarios involving stateless operations, concurrent processing, complex data transformations, immutability, mathematical problems, DSL creation, data analysis, testing, and reactive programming.

3. How can concurrency improve software performance?

By boosting throughput, optimizing resource usage, improving responsiveness, decreasing latency, enabling parallel processing, increasing I/O efficiency, facilitating scalability, and permitting task prioritization, concurrency enhances software performance and eventually results in faster and more effective applications.