Assignment-13

NAME:T.Akanksha

Hall ticket no:2403A510D8

Course:AI-Assisted coding

Branch:cse AIML

Batch_01

QUESTION

Task Description #1 – Remove Repetition

Task: Provide AI with the following redundant code and ask it to refactor

Python Code

```
def calculate_area(shape, x, y=0):
    if shape == "rectangle":
        return x * y
    elif shape == "square":
        return x * x
    elif shape == "circle":
        return 3.14 * x * x
```

Expected Output

- Refactored version with dictionary-based dispatch or separa functions.
- Cleaner and modular design.

Task Description #2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

Python Code

```
def read_file(filename):
```

Task Description #3 – Complex Refactoring

Task: Provide this legacy class to AI for readability and modularity improvements:

Python Code

```
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

Expected Output:

- AI improves naming (`name`, `age`, `marks`).
- Adds docstrings.
- Improves print readability.
- Possibly uses `sum(self.marks)` if marks stored in a list.

```
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

xpected Output:

- AI improves naming (`name`, `age`, `marks`).
- Adds docstrings.
- Improves print readability.
- Possibly uses `sum(self.marks)` if marks stored in a list.

Task Description #4 – Inefficient Loop Refactoring

Task: Refactor this inefficient loop with AI help

Python Code

```
nums = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in nums:
    squares.append(i * i)
```

xpected Output: AI suggested a **list comprehension**

TASK_1

```python
import math

def calculate_area(shape, x, y=0):
  """
  Calculates the area of different shapes.

  Args:
    shape: The shape type ('rectangle', 'square', 'circle').
    x: The primary dimension (side for square, radius for circle, length for rectangle).
    y: The secondary dimension for a rectangle (width). Defaults to 0.

  Returns:
    The calculated area of the shape, or "Invalid shape" if the shape is not recognized.
  """
  area_calculators = {
      "rectangle": lambda l, w: l * w,
      "square": lambda s: s * s,
      "circle": lambda r: math.pi * r * r
  }

  if shape in area_calculators:
    if shape == "rectangle":
      return area_calculators[shape](x, y)
    else:
      return area_calculators[shape](x)
  else:
    return "Invalid shape"
```

```python
  }

  if shape in area_calculators:
    if shape == "rectangle":
      return area_calculators[shape](x, y)
    else:
      return area_calculators[shape](x)
  else:
    return "Invalid shape"

# Example usage:
print(f"Area of a square with side 5: {calculate_area('square', 5)}")
print(f"Area of a rectangle with length 4 and width 6: {calculate_area('rectangle', 4, 6)}")
print(f"Area of a circle with radius 3: {calculate_area('circle', 3)}")
print(f"Area of a triangle with base 4 and height 5: {calculate_area('triangle', 4, 5)}") # Example of u
```

```
Area of a square with side 5: 25
Area of a rectangle with length 4 and width 6: 24
Area of a circle with radius 3: 28.274333882308138
Area of a triangle with base 4 and height 5: Invalid shape
```

TASK_2

```python
def read_file(filename):
  """
  Reads the content of a file with error handling.

  Args:
    filename: The name of the file to read.

  Returns:
    The content of the file if successful, otherwise an error message.
  """
  try:
    with open(filename, 'r') as f:
      data = f.read()
      return data
  except FileNotFoundError:
    return f"Error: File '{filename}' not found."
  except IOError:
    return f"Error: Could not read file '{filename}'."
  except Exception as e:
    return f"An unexpected error occurred: {e}"

# Example usage:
# Create a dummy file for testing
with open("my_test_file.txt", "w") as f:
  f.write("This is some test content.")

print(read_file("my_test_file.txt"))
```

```python
print(read_file("my_test_file.txt"))
print(read_file("non_existent_file.txt"))
```

```
This is some test content.
Error: File 'non_existent_file.txt' not found.
```

TASK_3

```python
class Student:
    """
    Represents a student with name, age, and marks in three subjects.
    """
    def __init__(self, name, age, mark1, mark2, mark3):
        """
        Initializes a Student object.

        Args:
            name (str): The name of the student.
            age (int): The age of the student.
            mark1 (int): Mark in subject 1.
            mark2 (int): Mark in subject 2.
            mark3 (int): Mark in subject 3.
        """
        self.name = name
        self.age = age
        self.marks = [mark1, mark2, mark3]

    def details(self):
        """
        Prints the name and age of the student.
        """
        print(f"Name: {self.name}, Age: {self.age}")

    def total(self):


     def total(self):
         """
         Calculates and returns the total marks of the student.
         """
         return sum(self.marks)

# Example usage:
student1 = Student("Alice", 16, 85, 90, 78)
student1.details()
print(f"Total marks: {student1.total()}")
```

```
Name: Alice, Age: 16
Total marks: 253
```

TASK_4

```python
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [i * i for i in nums]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```