AI ASSISTED CODING 13-2

Name: Nareddy Jasmitha

Enrolment no : 2403a51281

Course : AI ASSISTED CODING

Specialization : CSE-AIML

Date: 28-10-2025

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
```

PROMPT:

Develop a python code to generate 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 separate functions.**
- **Cleaner and modular design**

```python
import math

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

    Args:
        shape (str): The shape for which to calculate the area ("rectangle", "square", or "circle").
        x (float): The primary dimension (side for square, radius for circle, length for rectangle).
        y (float, optional): The secondary dimension for a rectangle (width). Defaults to 0.

    Returns:
        float: The calculated area of the shape, or None if the shape is not recognized.
    """
    if shape == "rectangle":
        return x * y
    elif shape == "square":
        return x ** 2  # Using exponentiation for square
    elif shape == "circle":
        return math.pi * x ** 2 # Using math.pi for a more accurate value of pi

    return None # Return None for unrecognized shapes

# Example usage:
print(f"Area of rectangle (4, 5): {calculate_area('rectangle', 4, 5)}")
print(f"Area of square (5): {calculate_area('square', 5)}")
print(f"Area of circle (3): {calculate_area('circle', 3)}")
print(f"Area of triangle (invalid): {calculate_area('triangle', 3)}")
```

```
Area of rectangle (4, 5): 20
Area of square (5): 25
Area of circle (3): 28.274333882308138
Area of triangle (invalid): None
```

EXPLANATION;

- `import math`: Imports the `math` module to use `math.pi` for a more accurate value of pi when calculating the area of a circle.
- `def calculate_area(shape, x, y=0):`: Defines the function `calculate_area` which takes the `shape` as a string, `x` as the primary dimension, and `y` as an optional secondary dimension (defaulting to 0).
- `if shape == "rectangle": return x * y`: If the shape is "rectangle", it returns the product of `x` and `y`.
- `elif shape == "square": return x ** 2`: If the shape is "square", it returns the square of `x` (using exponentiation).
- `elif shape == "circle": return math.pi * x ** 2`: If the shape is "circle", it returns the area of a circle using `math.pi` and the square of `x` (radius).
- `return None`: If the shape is not recognized, the function returns `None`.
- The lines starting with `print` demonstrate how to use the function with different shapes and print the calculated areas.

Task Description #2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

Python Code

```
def read_file(filename):
f = open(filename, "r")
data = f.read()
f.close()
return data
```

PROMPT:

Generate a python code to develop task: Legacy function without proper error handling Python Code def read_file(filename): f = open(filename, "r") data = f.read() f.close() return data Expected Output: AI refactors with with open() and try-except:

Expected Output:
AI refactors with with open() and try-except:

```python
def read_file(filename):
  try:
    with open(filename, "r") as f:
      data = f.read()
    return data
  except FileNotFoundError:
    print(f"Error: File '{filename}' not found.")
    return None
  except Exception as e:
    print(f"An error occurred: {e}")
    return None
```

EXPLANATION:

- `def read_file(filename):` : This line defines the function `read_file` which accepts one argument, `filename`.
- `try:` : This block starts a `try` block, which allows you to test a block of code for errors.
- `with open(filename, "r") as f:` : This opens the file specified by `filename` in read mode (`"r"`). The `with` statement ensures that the file is automatically closed after the block is exited, even if errors occur. The opened file object is assigned to the variable `f`.
- `data = f.read()` : This reads the entire content of the file into the `data` variable.
- `return data` : If the file is read successfully, the function returns the `data`.
- `except FileNotFoundError:` : This block catches the `FileNotFoundError` specifically. If the specified file does not exist, this block is executed.
- `print(f"Error: File '{filename}' not found.")` : This prints an error message indicating that the file was not found.
- `return None` : If a `FileNotFoundError` occurs, the function

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

PROMPT:

Develop a code to generate 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

Expected Output:

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

```python
class Student:
    """
    Represents a student with their 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): The mark in the first subject.
            mark2 (int): The mark in the second subject.
            mark3 (int): The mark in the third subject.
        """
        self.name = name
        self.age = age
        self.marks = [mark1, mark2, mark3]

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

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

EXPLANATION:

- **`class Student:`**: This line defines the `Student` class.
- **`""" Represents a student with their name, age, and marks in three subjects. """`**: This is a docstring that provides a brief description of the class.
- **`def __init__(self, name, age, mark1, mark2, mark3):`**: This is the constructor of the class. It's called when you create a new `Student` object.
    - **`""" Initializes a Student object. ... """`**: This is a docstring for the constructor, explaining its purpose and arguments.
    - **`self.name = name`**: This line assigns the value of the `name` argument to the `name` attribute of the object.
    - **`self.age = age`**: This line assigns the value of the `age` argument to the `age` attribute of the object.
    - **`self.marks = [mark1, mark2, mark3]`**: This line creates a list containing the three marks and assigns it to the `marks` attribute of the object. Storing marks in a list makes it easier to work with them (e.g., calculate the sum).

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)

PROMPT:

evelop a python code to generate 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)

Expected Output: AI suggested a list comprehension

```
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]
```

EXPLANATION:

- **`nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`**: This line initializes a list named `nums` containing integers from 1 to 10.
- **`squares = [i * i for i in nums]`**: This is a list comprehension. It's a compact way to create lists.
    - **`i * i`**: This is the expression that is evaluated for each item in the iterable. In this case, it squares the current item (`i`).
    - **`for i in nums`**: This is the loop that iterates over each element in the `nums` list. For each iteration, the current element is assigned to the variable `i`.
    - The entire expression within the square brackets `[]` creates a new list where each element is the result of squaring the corresponding element from the `nums` list.
- **`print(squares)`**: This line prints the `squares` list to the console.