

Assignment-13.3

NAME: P. UMESH REDDY

H-T NO: 2403A510F9

BATCH NO: 06

COURSE: AI ASSISTED CODING

- 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 SEPARATE FUNCTIONS.

- CLEANER AND MODULAR DESIGN

- #PROMPT: REFACTOR THE FOLLOWING REDUNDANT CODE TO REMOVE REPETITION. USE A DICTIONARY-BASED DISPATCH OR SEPARATE FUNCTIONS FOR EACH SHAPE TO MAKE THE DESIGN CLEANER AND MORE MODULAR.

• CODE & OUTPUT:

```
13.3-01.py > ...
1 def calculate_area(shape, x, y=0):
2     if shape == "rectangle":
3         return x * y
4     elif shape == "square":
5         return x * x
6     elif shape == "circle":
7         return 3.14 * x * x
8     else:
9         raise ValueError("Unknown shape")
10 # Example usage:
11 print(calculate_area("rectangle", 4, 5))
12 print(calculate_area("square", 4))
13 print(calculate_area("circle", 3))
14 # Output / Test cases
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/
.py
20
16
28.259999999999998
```

• REFACTORED CODE & OUTPUT:

```
13.3-1.py > ...
1 def rectangle_area(x, y):
2     return x * y
3 def square_area(x, _=0):
4     return x * x
5 def circle_area(x, _=0):
6     return 3.14 * x * x
7 area_dispatch = {
8     "rectangle": rectangle_area,
9     "square": square_area,
10    "circle": circle_area
11 }
12 def calculate_area(shape, x, y=0):
13     if shape not in area_dispatch:
14         raise ValueError("Unknown shape")
15     return area_dispatch[shape](x, y)
16 # Example usage:
17 print(calculate_area("rectangle", 4, 5))
18 print(calculate_area("square", 4))
19 print(calculate_area("circle", 3))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/
20
16
28.259999999999998
```

- **EXPLANATION FOR THE ORIGINAL CODE:**

THE FUNCTION CALCULATE_AREA COMPUTES THE AREA FOR THREE DIFFERENT SHAPES BASED ON THE SHAPE ARGUMENT:

- IF THE SHAPE IS "RECTANGLE", IT MULTIPLIES X AND Y TO GET THE AREA.
- IF THE SHAPE IS "SQUARE", IT RETURNS X * X (AREA OF A SQUARE).
- IF THE SHAPE IS "CIRCLE", IT RETURNS $3.14 * X * X$ (AREA OF A CIRCLE WITH RADIUS X).
- IF THE SHAPE IS NOT RECOGNIZED, IT RAISES A VALUEERROR.

EXAMPLE USAGE:

- CALCULATE_AREA("RECTANGLE", 4, 5) RETURNS 20
- CALCULATE_AREA("SQUARE", 4) RETURNS 16
- CALCULATE_AREA("CIRCLE", 3) RETURNS 28.26
- **EXPLANATION FOR THE REFACTORED CODE (FROM PREVIOUS ANSWER):**

THE REFACTORED VERSION SEPARATES THE AREA CALCULATION FOR EACH SHAPE INTO ITS OWN FUNCTION AND USES A DICTIONARY (AREA_DISPATCH) TO MAP SHAPE NAMES TO THESE FUNCTIONS. THE MAIN CALCULATE_AREA FUNCTION LOOKS UP THE APPROPRIATE FUNCTION AND CALLS IT.

BENEFITS OF THE REFACTORED APPROACH:

- **MODULARITY:** EACH SHAPE'S LOGIC IS ISOLATED, MAKING THE CODE EASIER TO MAINTAIN AND EXTEND.
- **READABILITY:** THE MAIN FUNCTION IS CLEANER AND EASIER TO UNDERSTAND.
- **EXTENSIBILITY:** ADDING NEW SHAPES ONLY REQUIRES DEFINING A NEW FUNCTION AND UPDATING THE DICTIONARY.

BOTH VERSIONS CORRECTLY CALCULATE AREAS, BUT THE REFACTORED VERSION IS MORE MAINTAINABLE AND SCALABLE FOR LARGER APPLICATIONS.

- 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
```

EXPECTED OUTPUT:

AI REFACTORS WITH WITH OPEN() AND TRY-EXCEPT:

- #PROMPT: REFACTOR THE FOLLOWING LEGACY FUNCTION TO INCLUDE PROPER ERROR HANDLING AND RESOURCE MANAGEMENT USING WITH OPEN() AND A TRY-EXCEPT BLOCK.
- CODE & OUTPUT:

```

13.3-02.py > read_file
1 def read_file(filename):
2     """
3     Reads the contents of a file safely with error handling.
4     Args:
5     |     filename (str): The path to the file.
6     Returns:
7     |     str: The file contents, or an error message if reading fails.
8     """
9     try:
10        with open(filename, "r") as f:
11            data = f.read()
12            return data
13    except FileNotFoundError:
14        return "Error: File not found."
15    except IOError as e:
16        return f"I/OError: {e}"
17
18 # Example usage:
19 print(read_file("sample.txt"))      # Output: Hello, world!
20 print(read_file("missing.txt"))    # Output: Error: File not found.

```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

Python

+

▾

🗑

🗑

⋮

```

PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/Desktop/AIAC/13.3-02.py
Error: File not found.
Error: File not found.

```

- EXPLANATION:
- USES WITH OPEN() FOR AUTOMATIC FILE CLOSING, EVEN IF AN ERROR OCCURS.
- HANDLES FILENOTFOUNDERROR AND GENERAL IOERROR WITH USER-FRIENDLY MESSAGES.
- RETURNS THE FILE CONTENTS IF SUCCESSFUL, OR AN ERROR MESSAGE OTHERWISE.
- THIS APPROACH IS SAFER AND MORE ROBUST THAN THE LEGACY CODE

- 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.

- #PROMPT: REFACTOR THE FOLLOWING LEGACY STUDENT CLASS TO IMPROVE READABILITY AND MODULARITY. USE DESCRIPTIVE VARIABLE NAMES, ADD DOCSTRINGS, IMPROVE PRINT FORMATTING, AND CONSIDER STORING MARKS IN A LIST TO SIMPLIFY CALCULATIONS.

- CODE & OUTPUT:

```
13.3-3.py > ...
1 class Student:
2     def __init__(self, n, a, m1, m2, m3):
3         self.n = n
4         self.a = a
5         self.m1 = m1
6         self.m2 = m2
7         self.m3 = m3
8     def details(self):
9         print("Name:", self.n, "Age:", self.a)
10    def total(self):
11        return self.m1+self.m2+self.m3
12    def average(self):
13        return (self.m1+self.m2+self.m3)/3
14 s1 = Student("Alice", 20, 85, 90, 95)
15 s1.details() # Output: Name: Alice Age: 20
16 print("Total Marks:", s1.total()) # Output: Total Marks: 270
17 print("Average Marks:", s1.average()) # Output: Average Marks: 90.0
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS Python + ~

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/top/AIAC/13.3-3.py
Name: Alice Age: 20
Total Marks: 270
Average Marks: 90.0
```

• REFACTORED CODE & OUTPUT:

```

1 class Student:
2     """
3     Represents a student with name, age, and a list of marks.
4     """
5
6     def __init__(self, name, age, marks):
7         """
8         Initialize a Student object.
9
10        Args:
11            name (str): The student's name.
12            age (int): The student's age.
13            marks (list of int): List of marks for the student.
14        """
15        self.name = name
16        self.age = age
17        self.marks = marks
18
19    def details(self):
20        """Print the student's name and age."""
21        print(f"Name: {self.name}, Age: {self.age}")
22
23    def total(self):
24        """Return the total of all marks."""
25        return sum(self.marks)
26
27 # Example usage and output
28 student = Student("Alice", 20, [85, 90, 95])
29 student.details()           # Output: Name: Alice, Age: 20
30 print(student.total())      # Output: 270

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python +

Name: Alice, Age: 20
270

EXPLANATION:

- VARIABLE NAMES ARE NOW DESCRIPTIVE (NAME, AGE, MARKS).
- MARKS ARE STORED AS A LIST, ALLOWING EASY CALCULATION WITH SUM(SELF.MARKS).
- DOCSTRINGS ARE ADDED FOR THE CLASS AND METHODS.
- THE DETAILS METHOD USES F-STRINGS FOR CLEARER OUTPUT.
- THE CODE IS NOW MORE MODULAR, READABLE, AND EASIER TO MAINTAIN OR EXTEND.

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)

EXPECTED OUTPUT: AI SUGGESTED A LIST COMPREHENSION

#PROMPT: REFACTOR THIS INEFFICIENT LOOP WITH AI HELP

CODE & OUTPUT:

```
1 # Refactored using list comprehension
2 nums = [1,2,3,4,5,6,7,8,9,10]
3 squares = [i * i for i in nums]
4 print(squares) # Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + - [] [] ... [] []

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/OneDrive/Desktop/AIAC/Untitled-2.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

EXPLANATION:

- THE ORIGINAL LOOP IS REPLACED WITH A LIST COMPREHENSION, WHICH IS MORE CONCISE AND EFFICIENT IN PYTHON.
- THIS APPROACH CREATES THE SQUARES LIST IN A SINGLE LINE, IMPROVING READABILITY AND PERFORMANCE FOR LARGE LISTS.