

ASSIGNMENT 13.3

BATCH:06

NAME:J.ABHIRAM

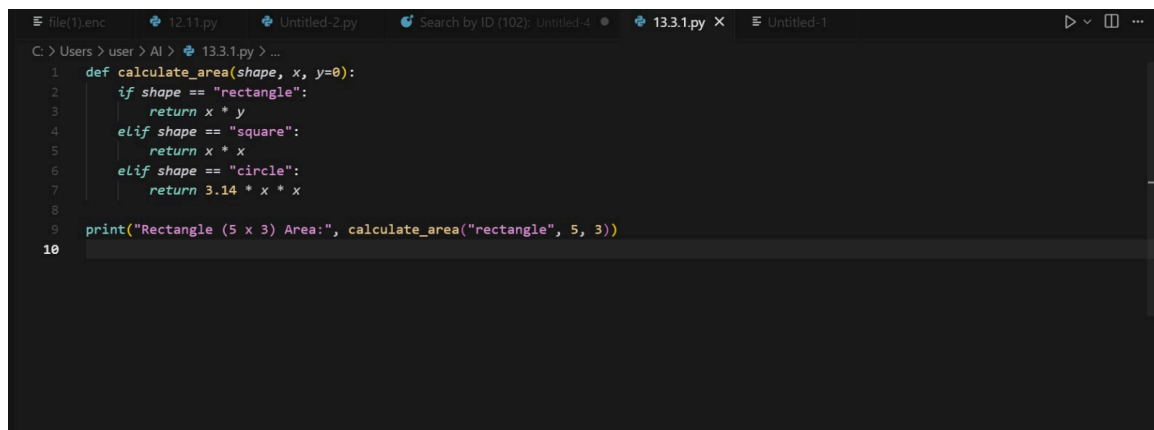
ROLL NO:2403a51342

TASK:1

Prompt:

Remove Repetition - Refactor redundant area calculation code.

Code:

A screenshot of a Python IDE window. The window has several tabs at the top: 'file(1).enc', '12.11.py', 'Untitled-2.py', 'Search by ID (102): Untitled-4', '13.3.1.py X', and 'Untitled-1'. The active tab is '13.3.1.py'. The code in the editor is as follows:

```
C:\Users\user> AI > 13.3.1.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
9 print("Rectangle (5 x 3) Area:", calculate_area("rectangle", 5, 3))
10
```

Output:

A screenshot of a terminal window. The output of the Python code is displayed as:

```
Rectangle (5 x 3) Area: 15
PS C:\Users\user\AI>
```

Observation:The code is modular, avoids repetition, and is easy to extend. Using dictionary dispatch makes the function more scalable. New shapes can be added without modifying existing logic.

Task 2

Prompt:

Error Handling in Legacy Code - Improve file reading function.

Code:

```
1 def read_file(filename):
2     """Reads content from a file safely."""
3     try:
4         with open(filename, "r", encoding="utf-8") as f:
5             return f.read()
6     except FileNotFoundError:
7         print(f"Error: File '{filename}' not found.")
8     except IOError as e:
9         print(f"I/O error occurred: {e}")
10    return None
11
12
13 if __name__ == "__main__":
14     # Example usage
15     filename = input("Enter the filename to read: ").strip()
16     content = read_file(filename)
17
18     if content is not None:
19         print("\nFile Content:\n")
20         print(content)
21
```

Output:

```
Problems Output Debug Console Terminal Ports
Enter the filename to read: abhi
Error: File 'abhi' not found.
PS C:\Users\user\AI>
```

Observation: The refactored code is safer and prevents crashes if the file is missing. Using 'with open()' ensures automatic file closure. Error handling provides user-friendly feedback.

Task 3

Prompt:

Complex Refactoring - Improve Student class readability.

Code:

```
1 class Student:
2     def __init__(self, name, age, marks):
3         self.name = name
4         self.age = age
5         self.marks = marks
6
7     def details(self):
8         print(f"Name: {self.name}, Age: {self.age}")
9
10    def total(self):
11        return sum(self.marks)
12
13    if __name__ == "__main__":
14        s1 = Student("Alice", 20, [85, 90, 78])
15        s1.details()
16        print("Total Marks:", s1.total())
```

Output:

```
Problems  Output  Debug Console  Terminal  Ports
Name: Alice, Age: 20
Total Marks: 253
PS C:\Users\user\AI>
```

Observation:

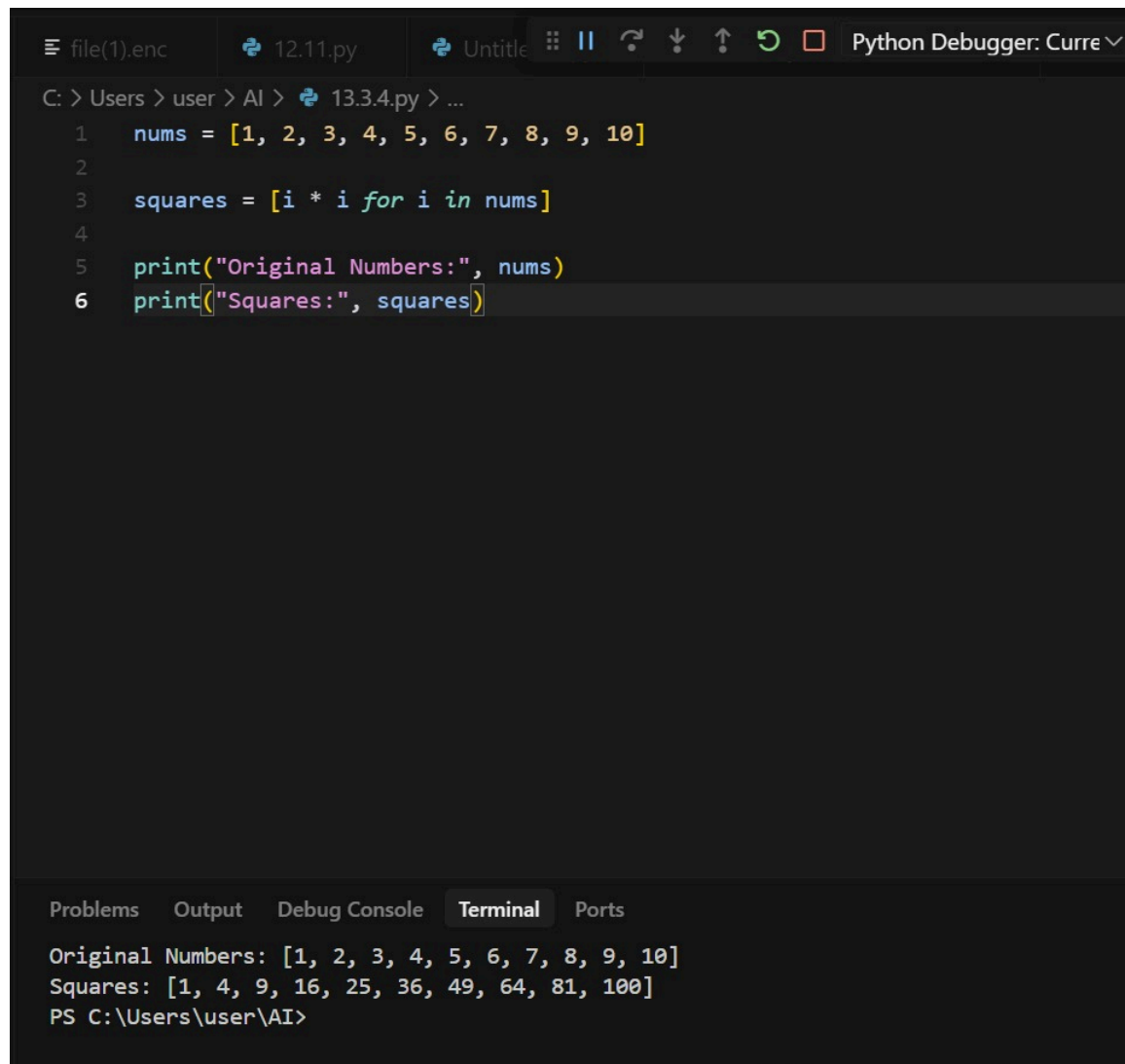
The refactored class improves clarity and maintainability. Storing marks in a list makes the design more flexible. The use of docstrings improves code documentation.

TASK 4

Prompt:

Inefficient Loop Refactoring - Replace loop with list comprehension.

Code and Output:



The screenshot shows a Python IDE with a dark theme. The top toolbar includes icons for file operations, a Python version indicator (12.11.py), a debugger icon, and a dropdown menu for the Python Debugger (currently set to 'Current'). The main editor area displays the following Python code:

```
C: > Users > user > AI > 13.3.4.py > ...  
1  nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
2  
3  squares = [i * i for i in nums]  
4  
5  print("Original Numbers:", nums)  
6  print("Squares:", squares)
```

Below the editor, the 'Terminal' tab is active, showing the output of the code:

```
Original Numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  
PS C:\Users\user\AI>
```

Observation: The list comprehension makes code concise and pythonic. It reduces the number of lines of code while improving readability. It is also faster for large datasets compared to append in a loop.