ASSIGNMENT 13.3

____BATCH:06

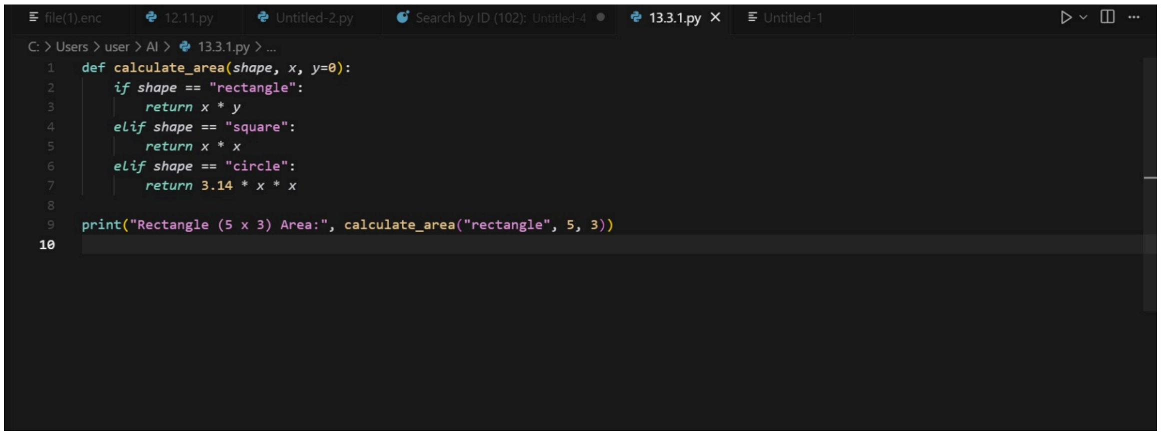
NAME:J.ABHIRAM
ROLL NO:2403a51342

TASK:1

Prompt:

Remove Repetition - Refactor redundant area calculation code.

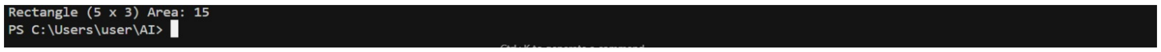
Code:



```python
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

print("Rectangle (5 x 3) Area:", calculate_area("rectangle", 5, 3))
```

Output:



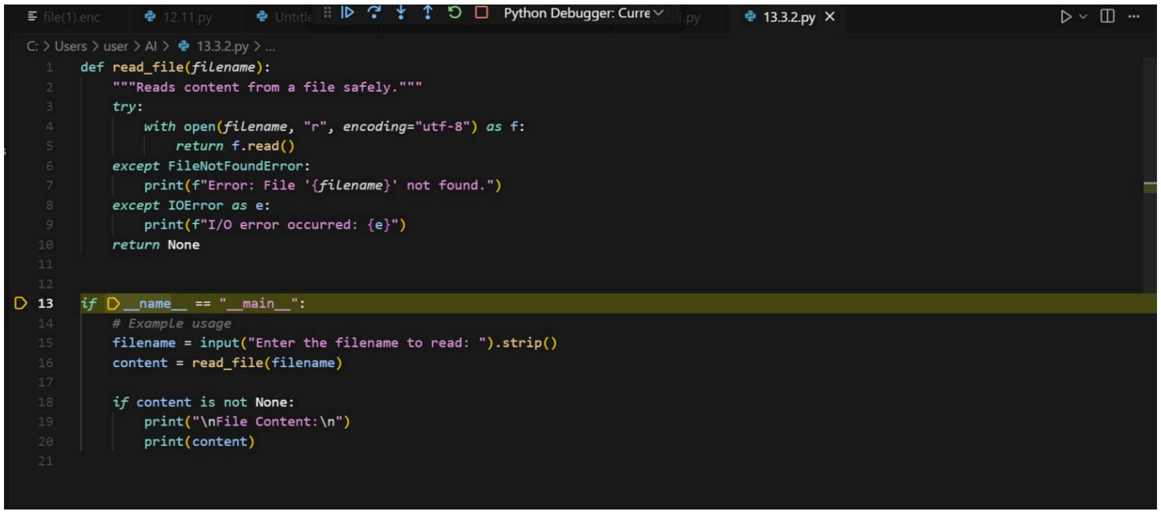
```
Rectangle (5 x 3) Area: 15
PS C:\Users\user\AI>
```

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:

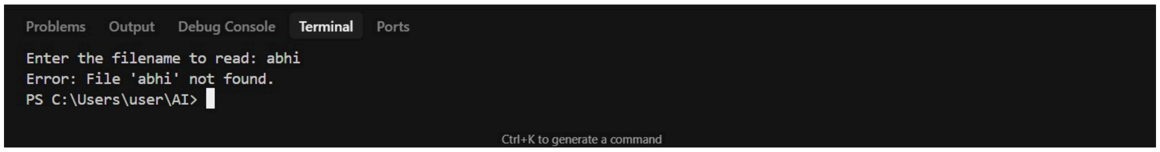


```python
def read_file(filename):
    """Reads content from a file safely."""
    try:
        with open(filename, "r", encoding="utf-8") as f:
            return f.read()
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
    except IOError as e:
        print(f"I/O error occurred: {e}")
    return None


if __name__ == "__main__":
    # Example usage
    filename = input("Enter the filename to read: ").strip()
    content = read_file(filename)

    if content is not None:
        print("\nFile Content:\n")
        print(content)
```

Output:



```
Enter the filename to read: abhi
Error: File 'abhi' not found.
PS C:\Users\user\AI>
```

Observation:

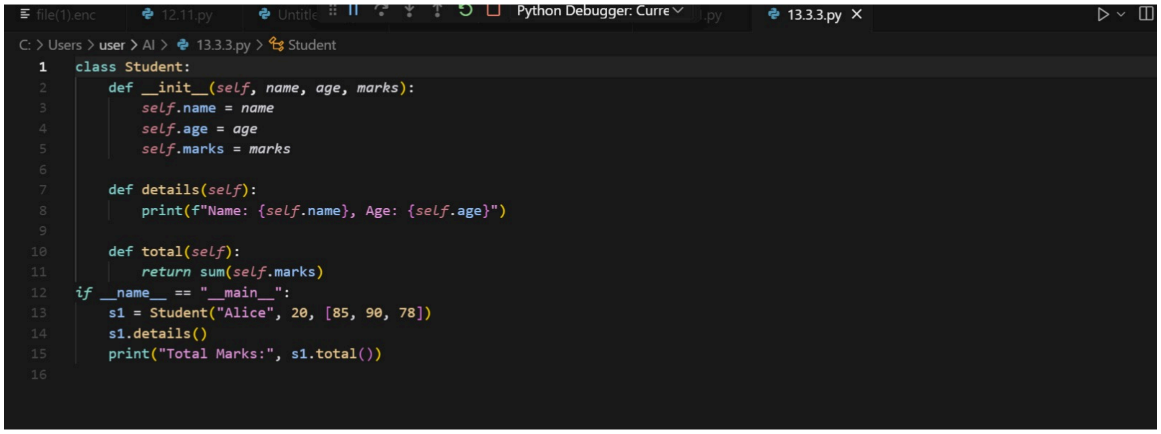
The refactored code is safer and prevents crashes if the file is missing. Using 'with open()' ensures automatic file closure. Error handling provides user-friendly feedback.

Task 3

Prompt:

Complex Refactoring - Improve Student class readability.

Code:

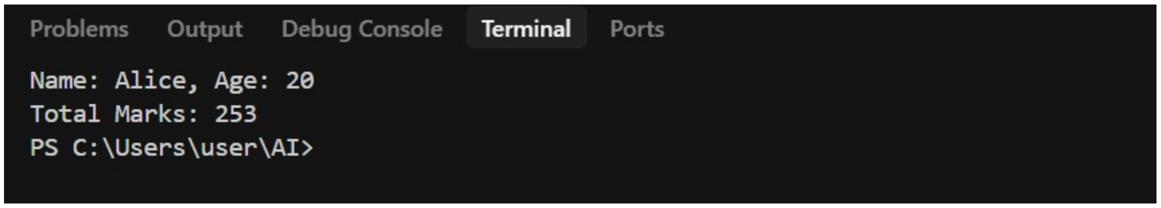


```python
class Student:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks

    def details(self):
        print(f"Name: {self.name}, Age: {self.age}")

    def total(self):
        return sum(self.marks)
if __name__ == "__main__":
    s1 = Student("Alice", 20, [85, 90, 78])
    s1.details()
    print("Total Marks:", s1.total())
```

Output:



```
Name: Alice, Age: 20
Total Marks: 253
PS C:\Users\user\AI>
```

Observation:

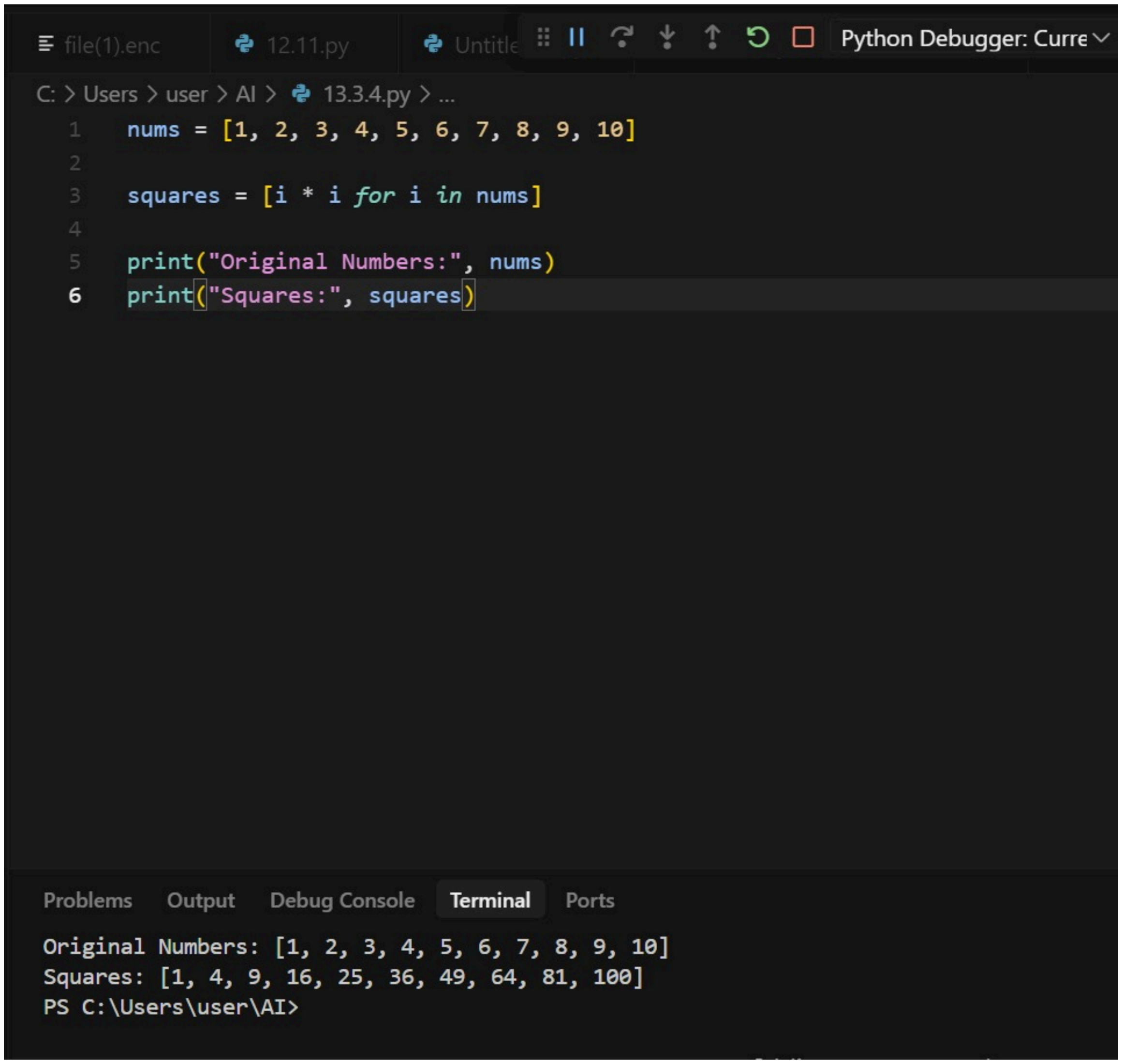
The refactored class improves clarity and maintainability. Storing marks in a list makes the design more flexible. The use of docstrings improves code documentation.

TASK 4

Prompt:

Inefficient Loop Refactoring - Replace loop with list comprehension.

Code and Output:



```python
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

squares = [i * i for i in nums]

print("Original Numbers:", nums)
print("Squares:", squares)
```

```
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.