Lab Assignment-10.1

Name:P.Chandra Vardhan Reddy

Hall ticket:2303A51034

Batch-01

Task Description #1 – Syntax and Logic Errors

Task: Use AI to identify and fix syntax and logic errors in a
faulty

Python script.

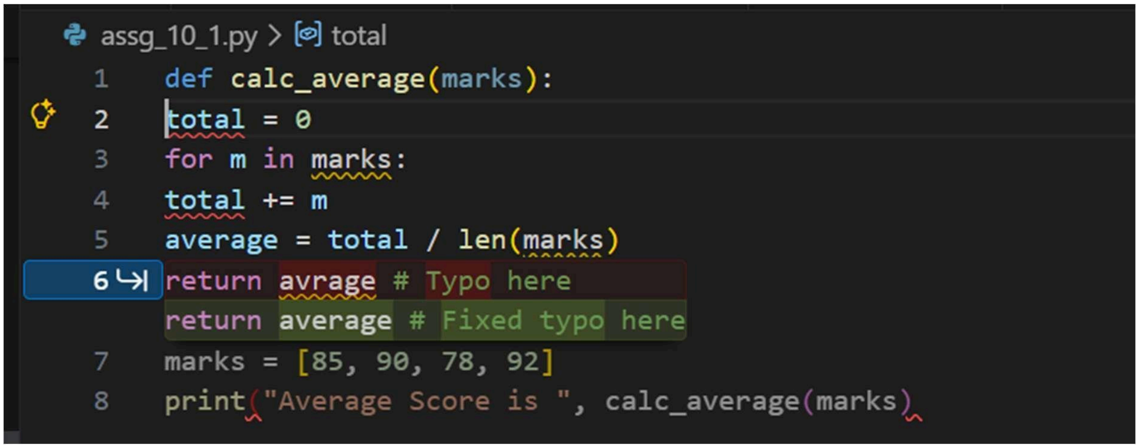
Sample Input Code:

```
# Calculate average score of a student
      def calc_average(marks):
            total = 0
            for m in marks:
                  total += m average = total
                  / len(marks)
      return avrage # Typo here marks = [85, 90,
      78, 92] print("Average Score is ",
      calc_average(marks)
```

Expected Output:

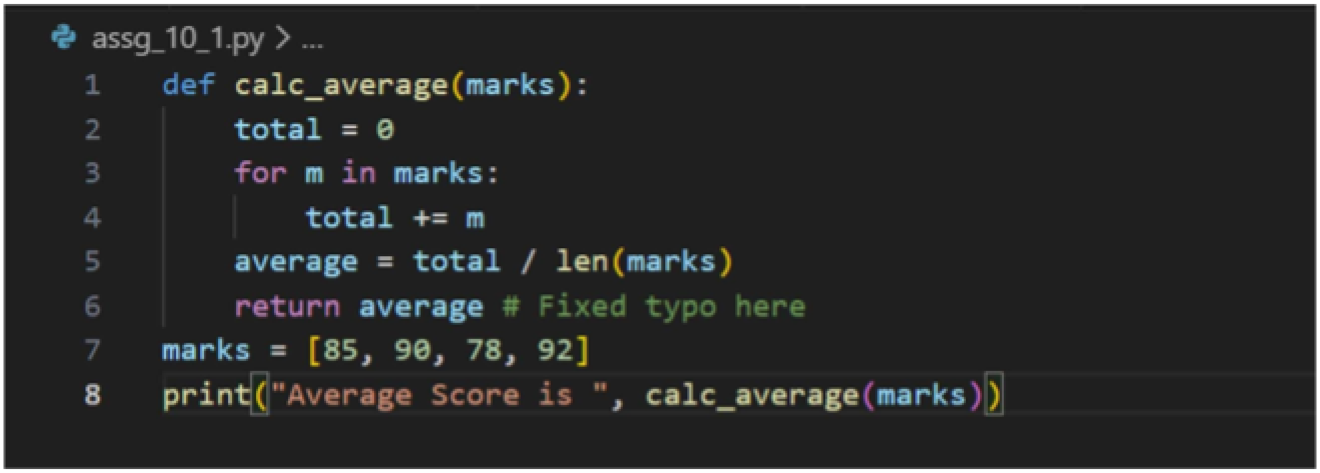
- Corrected and runnable Python code with explanations of the
  fixes.

Privous code:

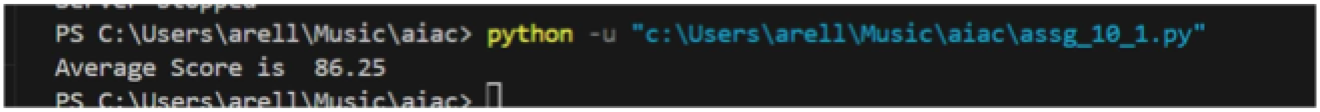


```python
def calc_average(marks):
total = 0
for m in marks:
total += m
average = total / len(marks)
return avrage # Typo here
return average # Fixed typo here
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks))
```

Corrected code:



```python
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return average # Fixed typo here
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks))
```



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
Average Score is  86.25
PS C:\Users\arell\Music\aiac> 
```

Task Description #2 – PEP 8 Compliance

Task: Use AI to refactor Python code to follow PEP 8 style guidelines.

Sample Input Code:

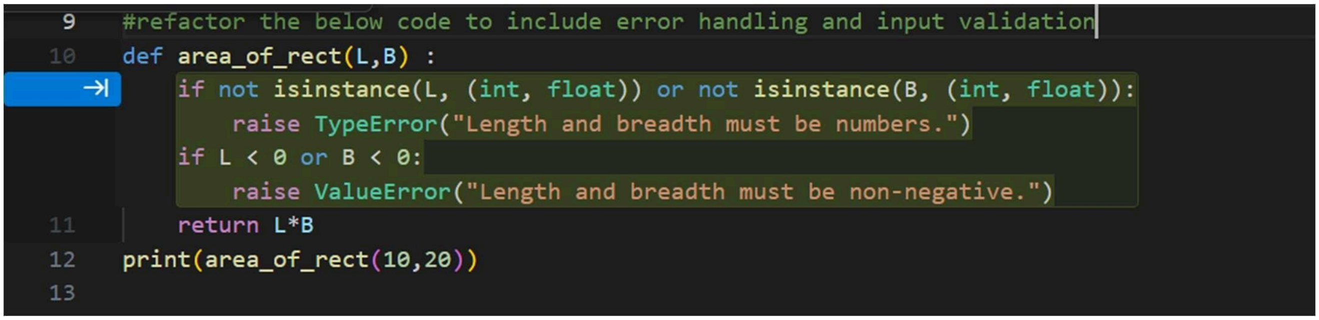
def area_of_rect(L,B) : return L*B

print(area_of_rect(10,20))

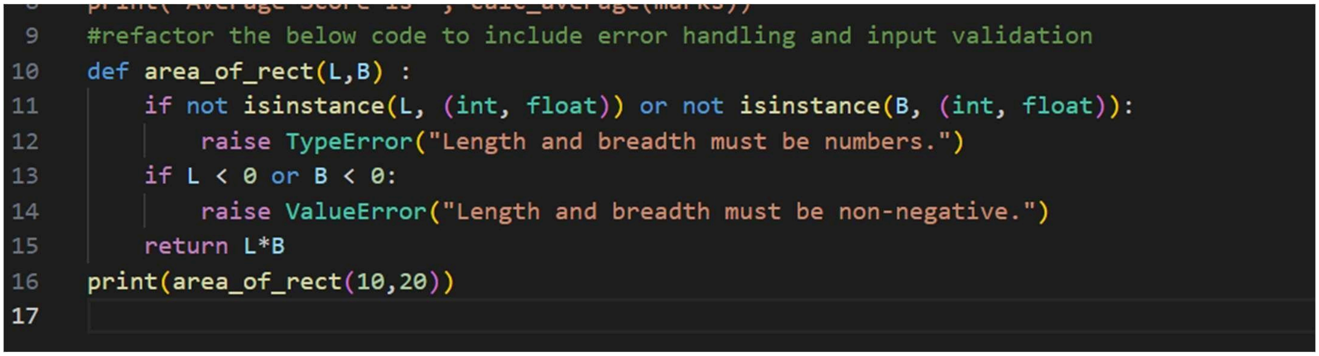
Expected Output:

- Well-formatted PEP 8-compliant Python code.

Code:

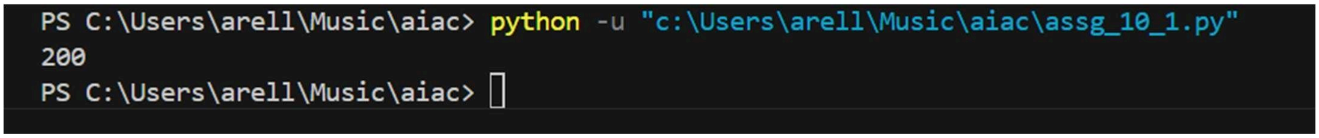


```python
#refactor the below code to include error handling and input validation
def area_of_rect(L,B) :
    if not isinstance(L, (int, float)) or not isinstance(B, (int, float)):
        raise TypeError("Length and breadth must be numbers.")
    if L < 0 or B < 0:
        raise ValueError("Length and breadth must be non-negative.")
    return L*B
print(area_of_rect(10,20))
```



```python
print("Average score is", calc_average(marks))
#refactor the below code to include error handling and input validation
def area_of_rect(L,B) :
    if not isinstance(L, (int, float)) or not isinstance(B, (int, float)):
        raise TypeError("Length and breadth must be numbers.")
    if L < 0 or B < 0:
        raise ValueError("Length and breadth must be non-negative.")
    return L*B
print(area_of_rect(10,20))
```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
200
PS C:\Users\arell\Music\aiac> 
```

Task Description #3 – Readability Enhancement

Task: Use AI to make code more readable without changing its logic.

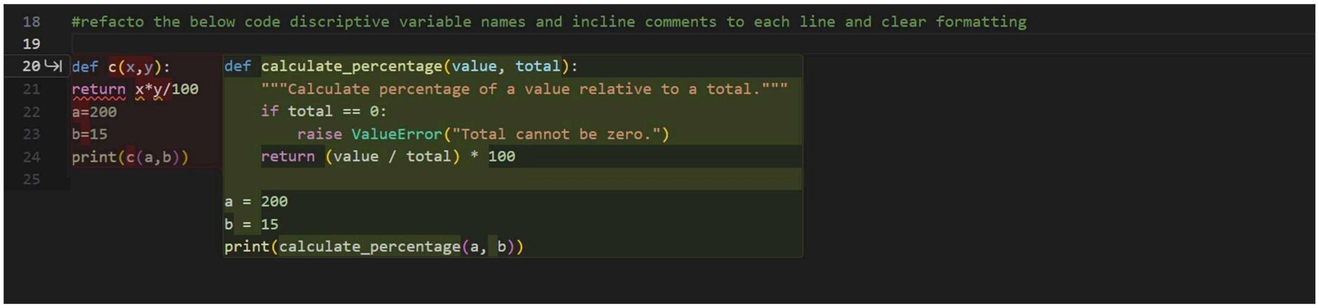
Sample Input Code:

```
def c(x,y): return
x*y/100 a=200 b=15
print(c(a,b))
```

Expected Output:

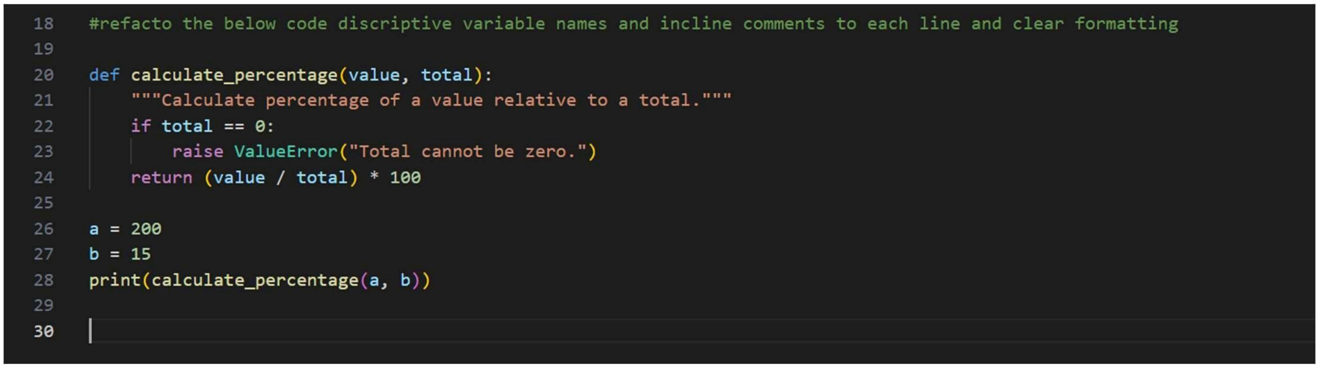
- Python code with descriptive variable names, inline comments, and clear formatting.

Code:



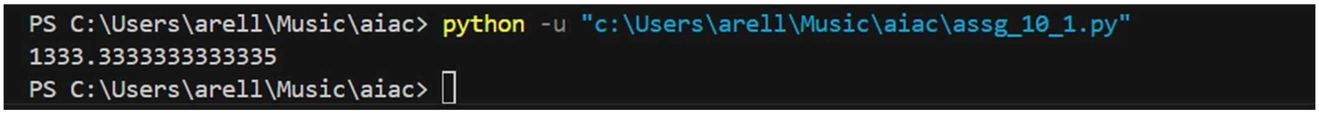
```
#refacto the below code discriptive variable names and incline comments to each line and clear formatting

def c(x,y):
return x*y/100
a=200
b=15
print(c(a,b))
```

```
def calculate_percentage(value, total):
    """Calculate percentage of a value relative to a total."""
    if total == 0:
        raise ValueError("Total cannot be zero.")
    return (value / total) * 100

a = 200
b = 15
print(calculate_percentage(a, b))
```



```
#refacto the below code discriptive variable names and incline comments to each line and clear formatting

def calculate_percentage(value, total):
    """Calculate percentage of a value relative to a total."""
    if total == 0:
        raise ValueError("Total cannot be zero.")
    return (value / total) * 100

a = 200
b = 15
print(calculate_percentage(a, b))

```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
1333.3333333333335
PS C:\Users\arell\Music\aiac> 
```

Task Description #4 – Refactoring for Maintainability

Task: Use AI to break repetitive or long code into reusable

functions.

Sample Input Code:

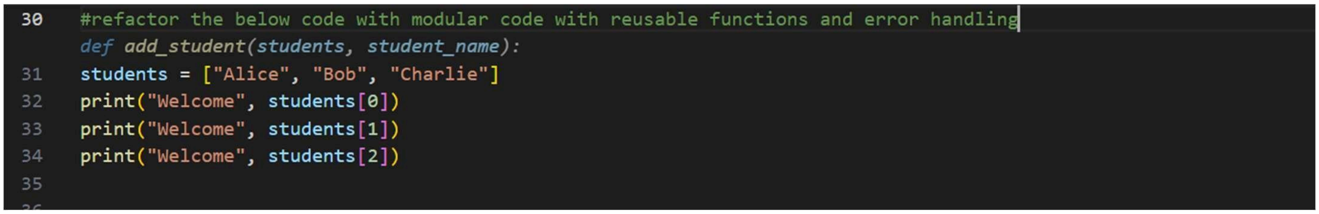
students = ["Alice", "Bob", "Charlie"]

print("Welcome", students[0]) print("Welcome",

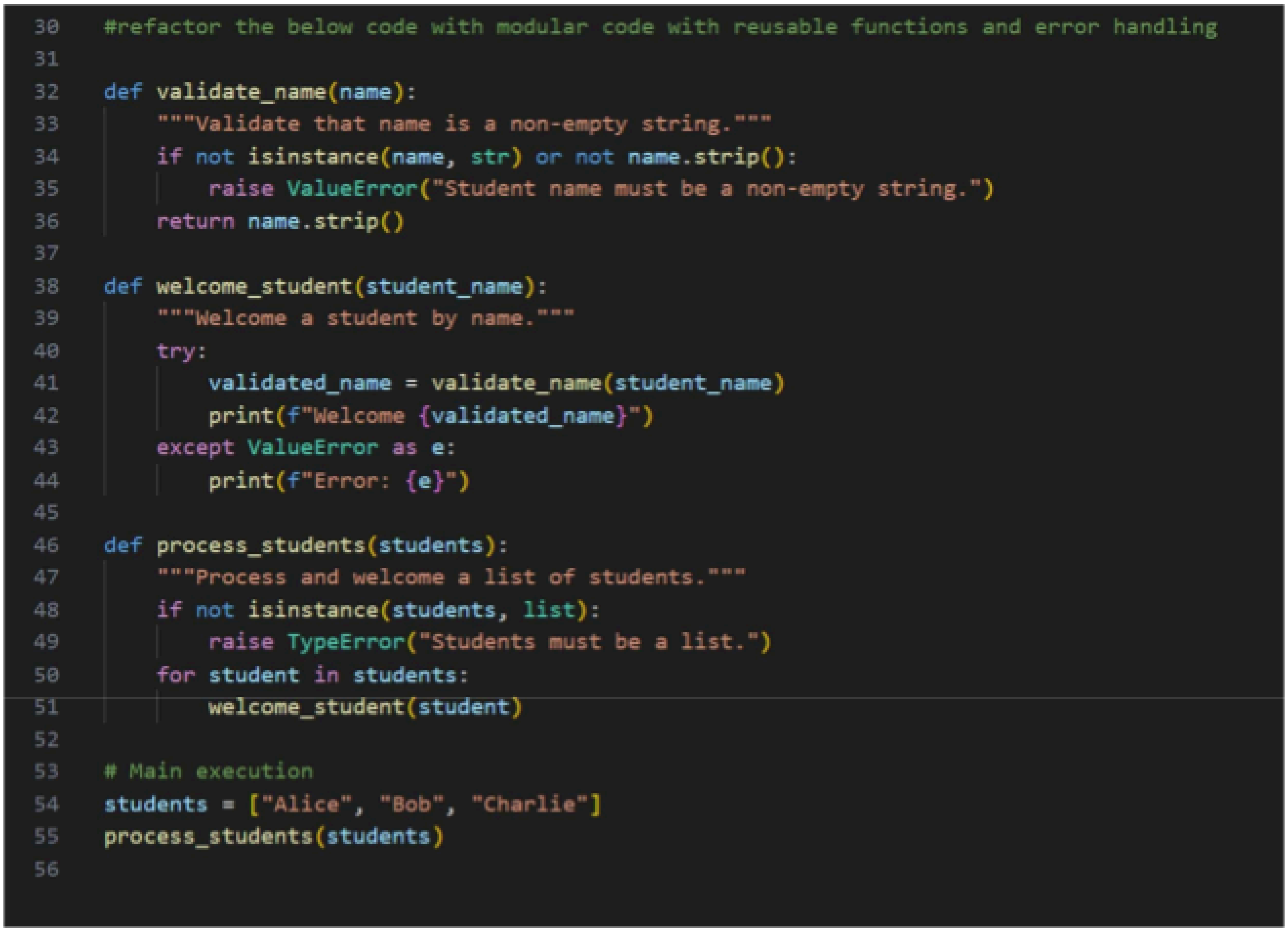
students[1]) print("Welcome", students[2])

Expected Output:

- Modular code with reusable functions.

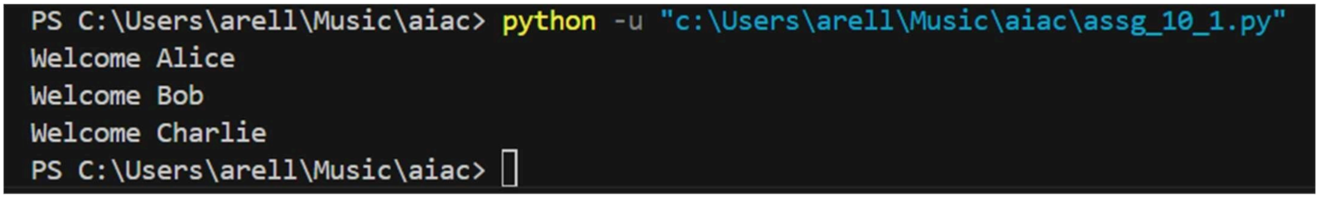


```
#refactor the below code with modular code with reusable functions and error handling
def add_student(students, student_name):
students = ["Alice", "Bob", "Charlie"]
print("Welcome", students[0])
print("Welcome", students[1])
print("Welcome", students[2])

```



```python
#refactor the below code with modular code with reusable functions and error handling

def validate_name(name):
    """Validate that name is a non-empty string."""
    if not isinstance(name, str) or not name.strip():
        raise ValueError("Student name must be a non-empty string.")
    return name.strip()

def welcome_student(student_name):
    """Welcome a student by name."""
    try:
        validated_name = validate_name(student_name)
        print(f"Welcome {validated_name}")
    except ValueError as e:
        print(f"Error: {e}")

def process_students(students):
    """Process and welcome a list of students."""
    if not isinstance(students, list):
        raise TypeError("Students must be a list.")
    for student in students:
        welcome_student(student)

# Main execution
students = ["Alice", "Bob", "Charlie"]
process_students(students)
```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
Welcome Alice
Welcome Bob
Welcome Charlie
PS C:\Users\arell\Music\aiac> 
```

Task Description #5 – Performance Optimization

Task: Use AI to make the code run faster.

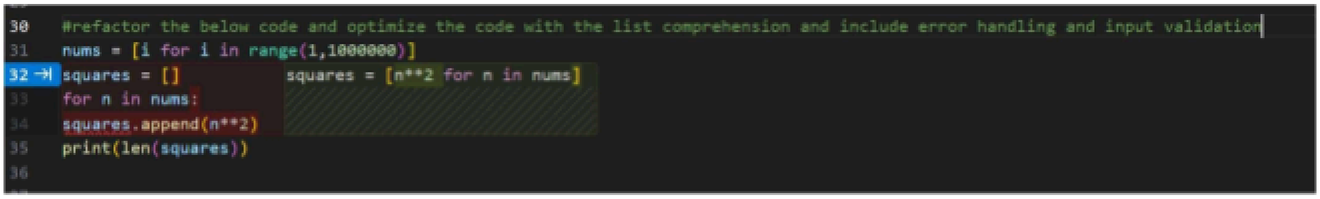
Sample Input Code:

```
# Find squares of numbers nums = [i for i
    in range(1,1000000)] squares = []
    for n in nums:
    squares.append(n**2)
    print(len(squares))
```

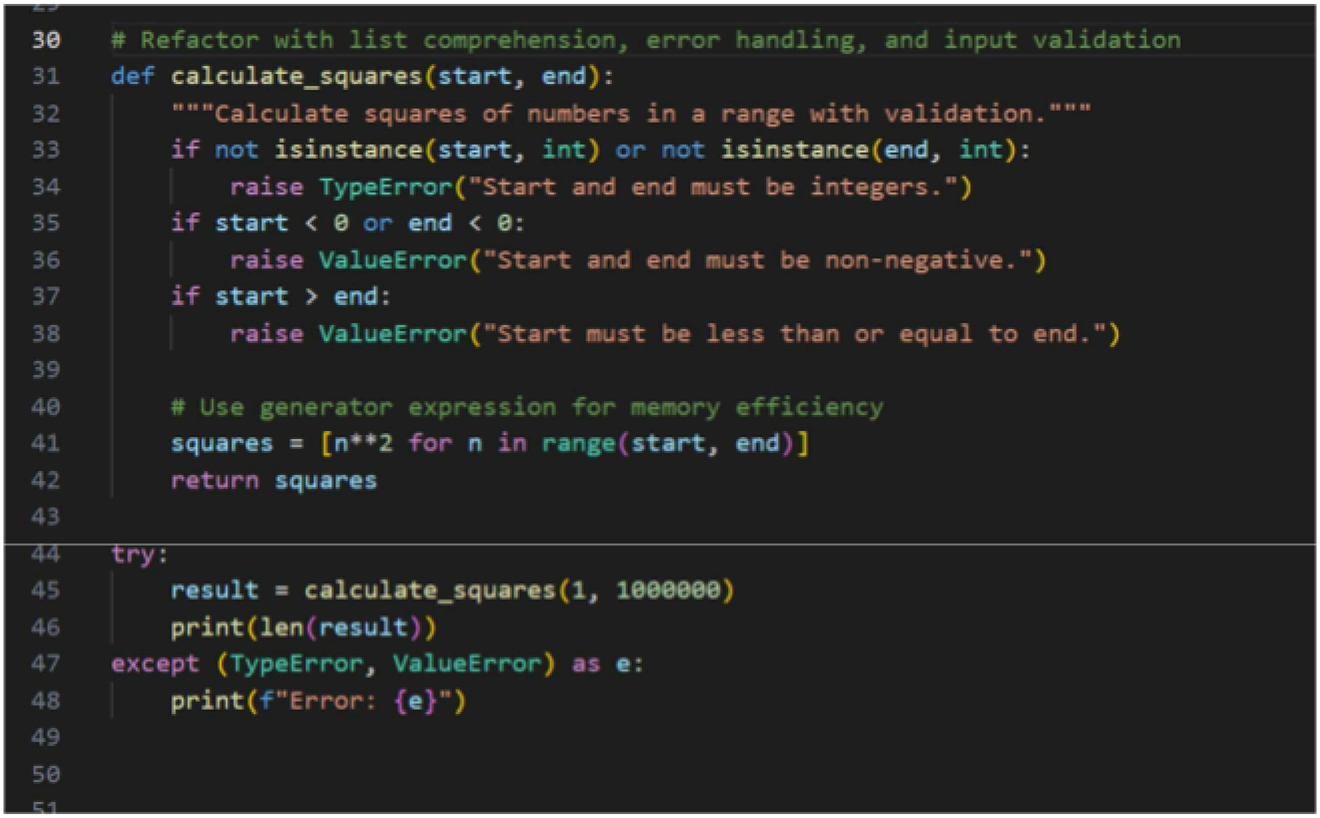
Expected Output:

- Optimized code using list comprehensions or vectorized operations.

Code:

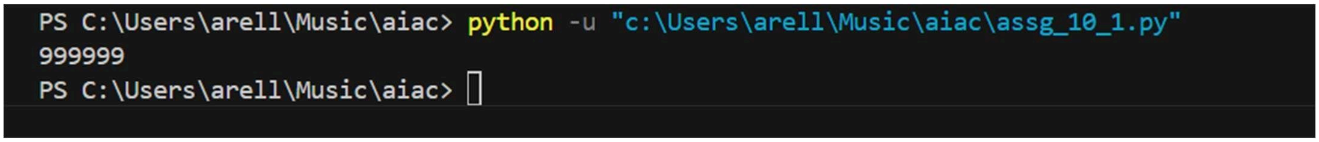


```
#refactor the below code and optimize the code with the list comprehension and include error handling and input validation
nums = [i for i in range(1,1000000)]
squares = []                    squares = [n**2 for n in nums]
for n in nums:
squares.append(n**2)
print(len(squares))
```



```
# Refactor with list comprehension, error handling, and input validation
def calculate_squares(start, end):
    """Calculate squares of numbers in a range with validation."""
    if not isinstance(start, int) or not isinstance(end, int):
        raise TypeError("Start and end must be integers.")
    if start < 0 or end < 0:
        raise ValueError("Start and end must be non-negative.")
    if start > end:
        raise ValueError("Start must be less than or equal to end.")

    # Use generator expression for memory efficiency
    squares = [n**2 for n in range(start, end)]
    return squares

try:
    result = calculate_squares(1, 1000000)
    print(len(result))
except (TypeError, ValueError) as e:
    print(f"Error: {e}")
```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
999999
PS C:\Users\arell\Music\aiac> 
```

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

Sample Input Code:

def grade(score):

if score >= 90:

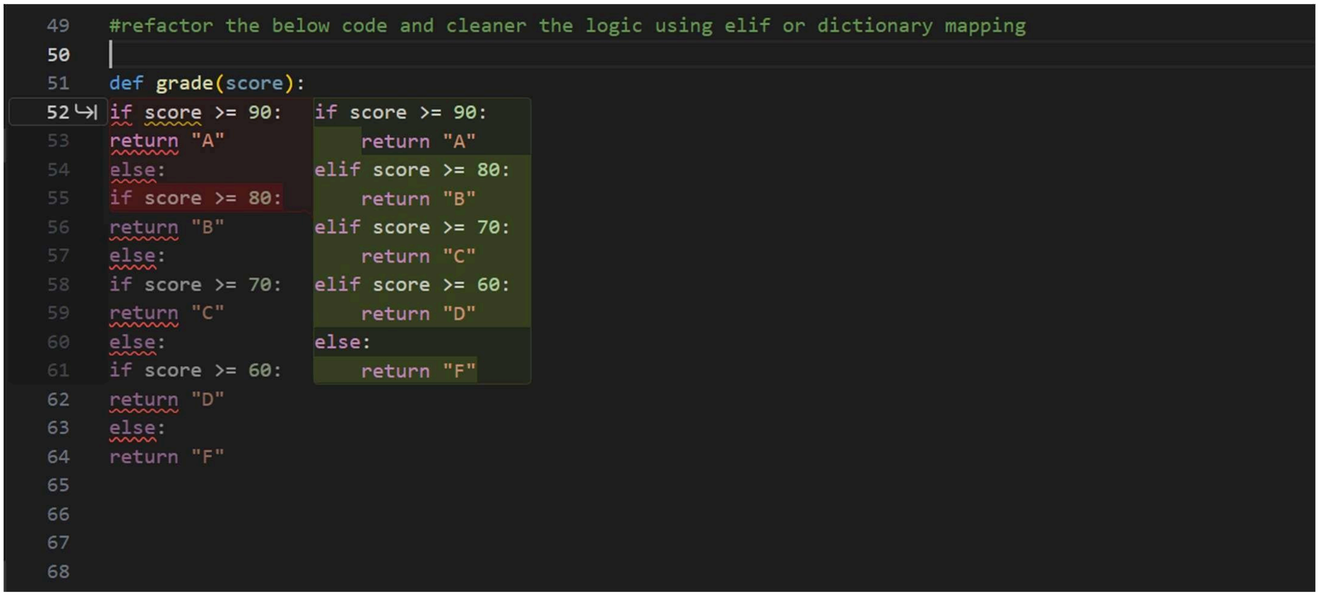
return "A" else:

```
if score >= 80:
return "B"
else:
if score >= 70:
return "C"
else:
if score >= 60:
return "D"
else:
return "F"
```

Expected Output:

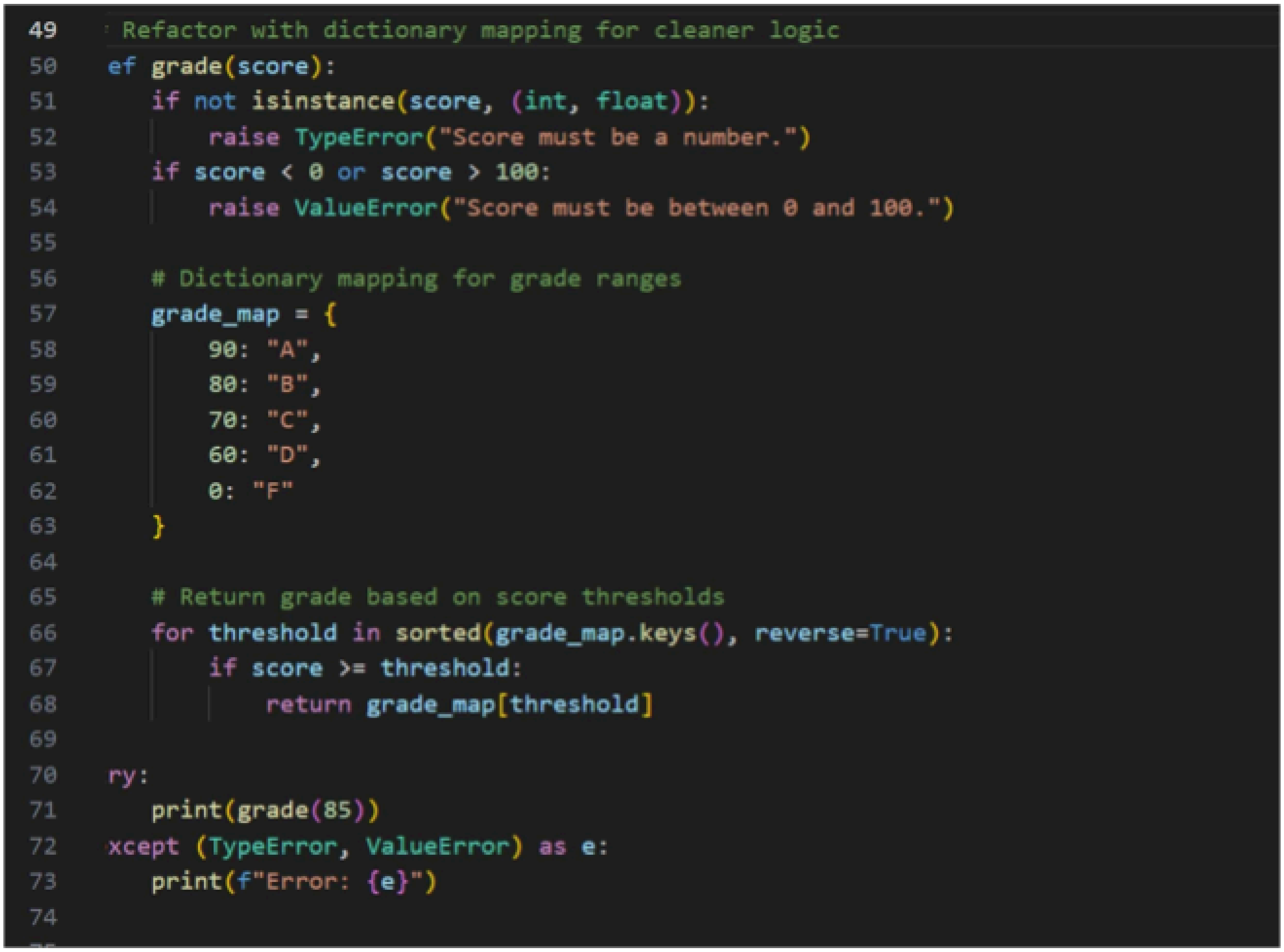
- Cleaner logic using elif or dictionary mapping.

Code:



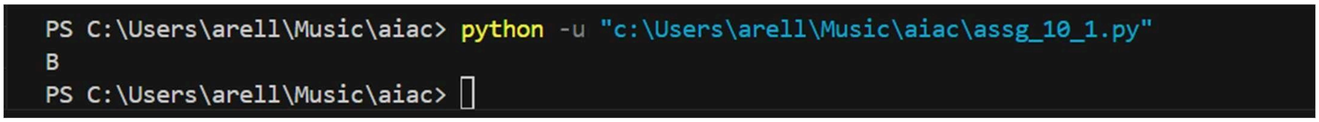
```
#refactor the below code and cleaner the logic using elif or dictionary mapping

def grade(score):
if score >= 90:
return "A"
else:
if score >= 80:
return "B"
else:
if score >= 70:
return "C"
else:
if score >= 60:
return "D"
else:
return "F"
```

```
if score >= 90:
    return "A"
elif score >= 80:
    return "B"
elif score >= 70:
    return "C"
elif score >= 60:
    return "D"
else:
    return "F"
```



```
 Refactor with dictionary mapping for cleaner logic
ef grade(score):
    if not isinstance(score, (int, float)):
        raise TypeError("Score must be a number.")
    if score < 0 or score > 100:
        raise ValueError("Score must be between 0 and 100.")

    # Dictionary mapping for grade ranges
    grade_map = {
        90: "A",
        80: "B",
        70: "C",
        60: "D",
        0: "F"
    }

    # Return grade based on score thresholds
    for threshold in sorted(grade_map.keys(), reverse=True):
        if score >= threshold:
            return grade_map[threshold]

ry:
    print(grade(85))
xcept (TypeError, ValueError) as e:
    print(f"Error: {e}")
```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
B
PS C:\Users\arell\Music\aiac> 
```