Assignment – 13.5

Name: Yashaswini Nagireddi
H.no: 2303A51418
Batch: 21

Task Description #1 (Refactoring – Removing Global Variables)

• Task: Use AI to eliminate unnecessary global variables from the code.

• Instructions:

o Identify global variables used across functions.
o Refactor the code to pass values using function parameters.
o Improve modularity and testability.

• Sample Legacy Code:

```
rate = 0.1
def calculate_interest(amount):
return amount * rate
print(calculate_interest(1000))
```

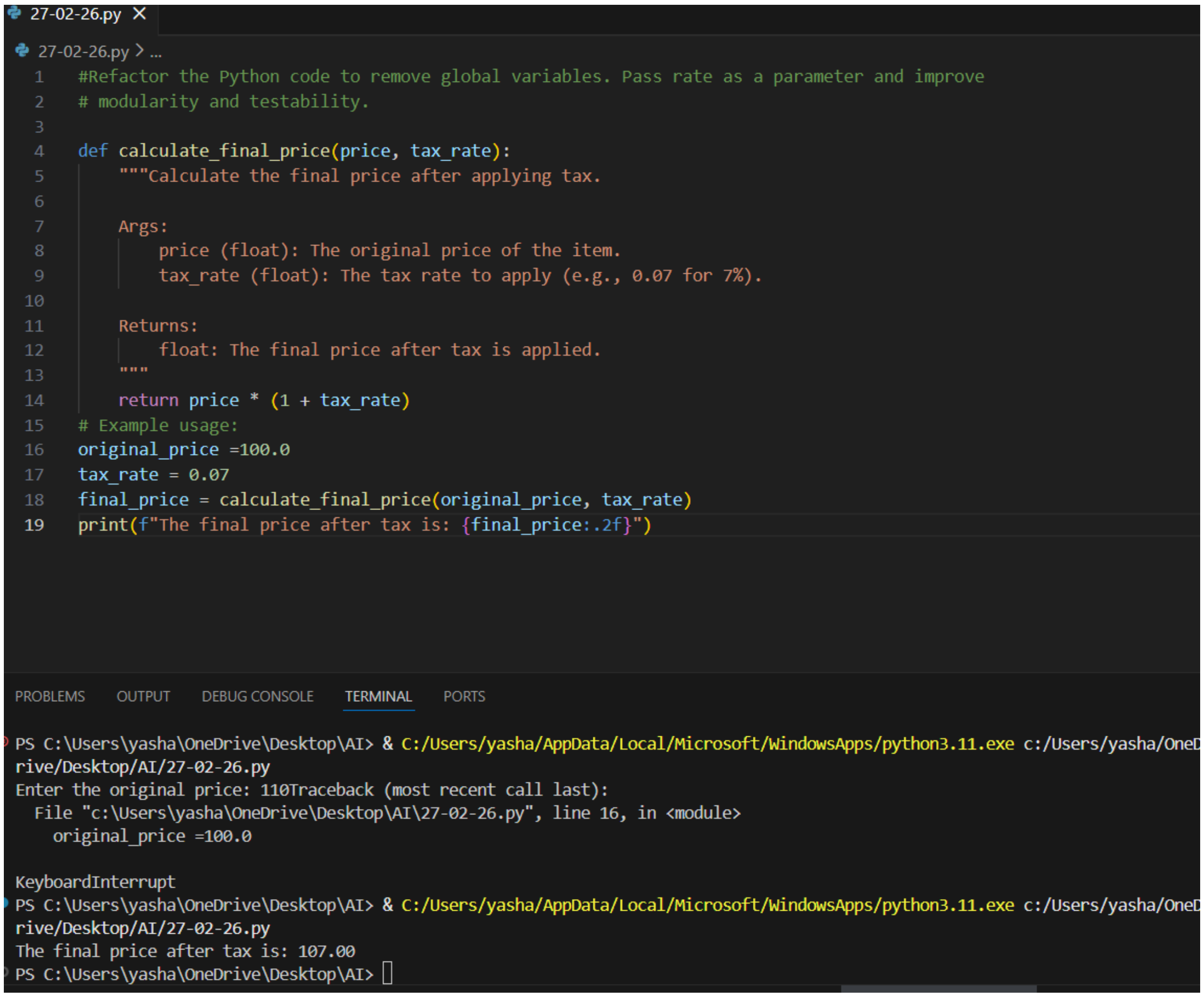
• Expected Output:

o Refactored version passing rate as a parameter or using a configuration structure.

Prompt:

Refactor the Python code to remove global variables. Pass rate as a parameter and improve modularity and testability.

Code:



```python
#Refactor the Python code to remove global variables. Pass rate as a parameter and improve
# modularity and testability.

def calculate_final_price(price, tax_rate):
    """Calculate the final price after applying tax.

    Args:
        price (float): The original price of the item.
        tax_rate (float): The tax rate to apply (e.g., 0.07 for 7%).

    Returns:
        float: The final price after tax is applied.
    """
    return price * (1 + tax_rate)
# Example usage:
original_price =100.0
tax_rate = 0.07
final_price = calculate_final_price(original_price, tax_rate)
print(f"The final price after tax is: {final_price:.2f}")
```

```
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
Enter the original price: 110Traceback (most recent call last):
  File "c:\Users\yasha\OneDrive\Desktop\AI\27-02-26.py", line 16, in <module>
    original_price =100.0

KeyboardInterrupt
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
The final price after tax is: 107.00
PS C:\Users\yasha\OneDrive\Desktop\AI>
```

Observation:

- Global variable removed.
- Function is now reusable and testable.
- Follows modular programming principles.

Task Description #2 : (Refactoring Deeply Nested Conditionals)

• Task: Use AI to refactor deeply nested if–elif–else logic into a cleaner structure.

• Focus Areas:

o Readability

o Logical simplification

o Maintainability

Legacy Code:

```
score = 78
if score >= 90:
print("Excellent")
else:
if score >= 75:
print("Very Good")
else:
if score >= 60:
print("Good")
else:
print("Needs Improvement")
```

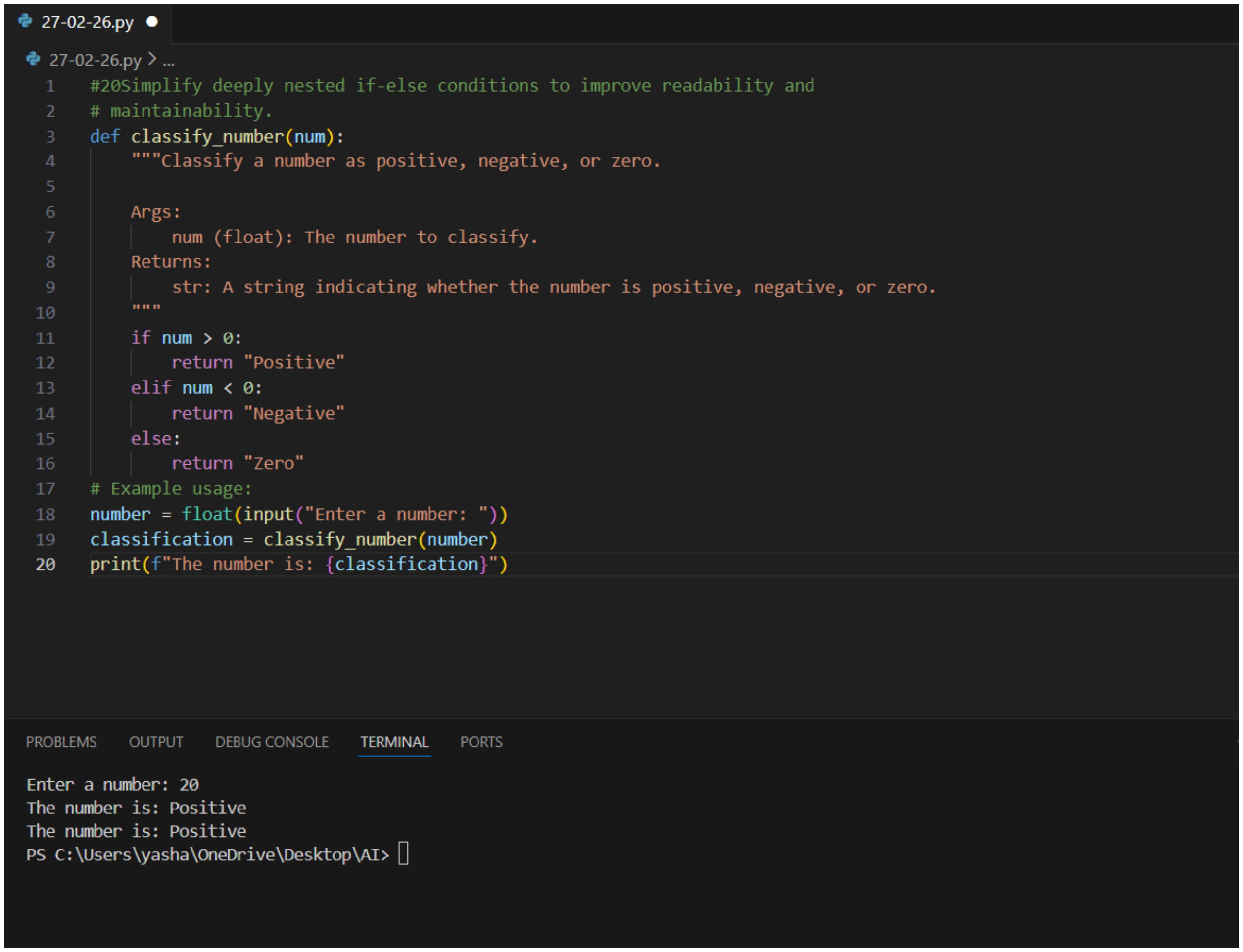
Expected Outcome:

o Flattened logic using guard clauses or a mapping-based approach.

Prompt:

Simplify deeply nested if-else conditions to improve readability and maintainability.

Code:



```python
#20Simplify deeply nested if-else conditions to improve readability and
# maintainability.
def classify_number(num):
    """Classify a number as positive, negative, or zero.

    Args:
        num (float): The number to classify.
    Returns:
        str: A string indicating whether the number is positive, negative, or zero.
    """
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    else:
        return "Zero"
# Example usage:
number = float(input("Enter a number: "))
classification = classify_number(number)
print(f"The number is: {classification}")
```

```
Enter a number: 20
The number is: Positive
The number is: Positive
PS C:\Users\yasha\OneDrive\Desktop\AI>
```

Observation:

- Removed nested structure.
- Improved readability.
- Easier to modify grade ranges.

Task 3 (Refactoring Repeated File Handling Code)

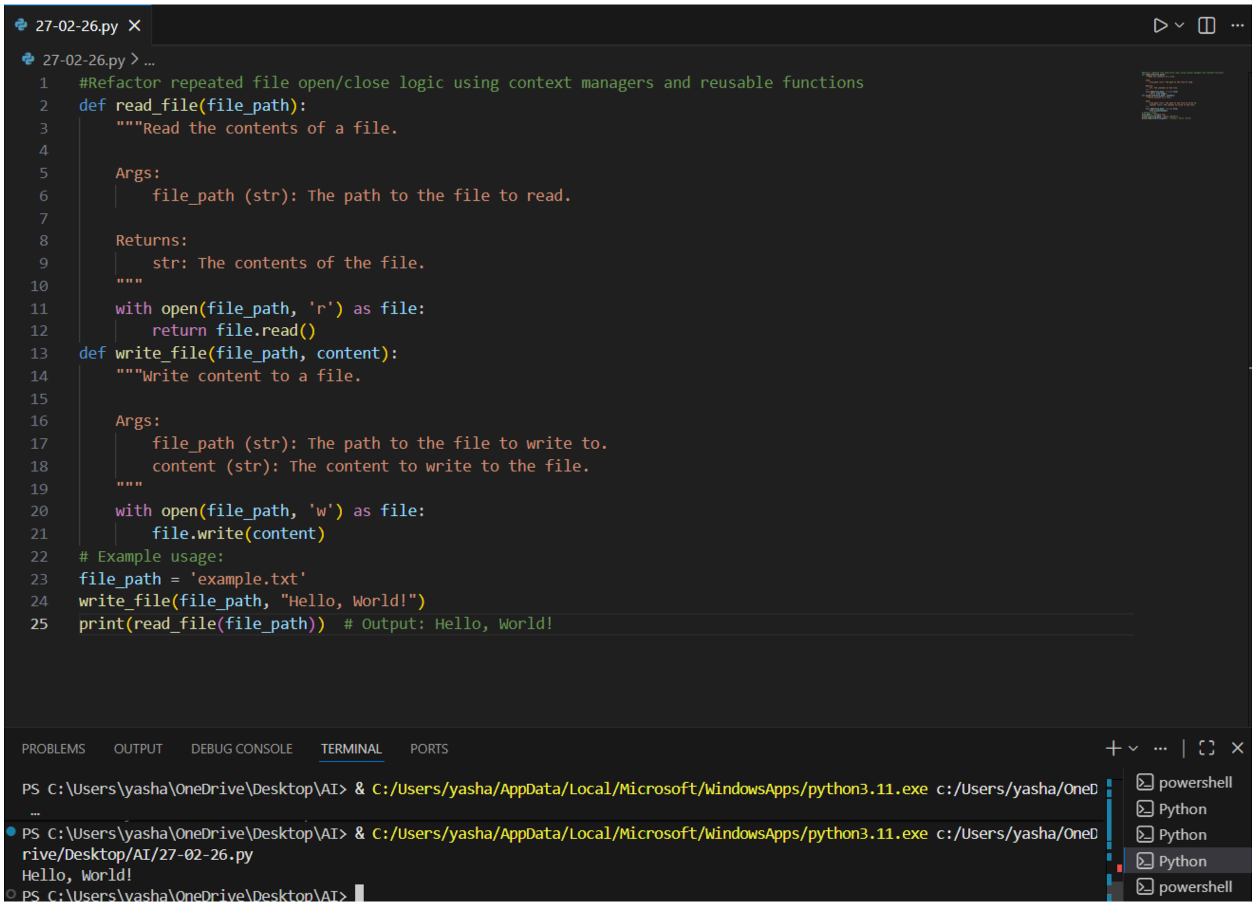
• Task: Use AI to refactor repeated file open/read/close logic.
• Focus Areas:
o DRY principle
o Context managers
o Function reuse
Legacy Code:
f = open("data1.txt")

print(f.read())
f.close()
f = open("data2.txt")
print(f.read())
f.close()
Expected Outcome:
o Reusable function using with open() and parameters.

Prompt:

Refactor repeated file open/close logic using context managers and reusable functions.

Code:



```
#Refactor repeated file open/close logic using context managers and reusable functions
def read_file(file_path):
    """Read the contents of a file.

    Args:
        file_path (str): The path to the file to read.

    Returns:
        str: The contents of the file.
    """
    with open(file_path, 'r') as file:
        return file.read()
def write_file(file_path, content):
    """Write content to a file.

    Args:
        file_path (str): The path to the file to write to.
        content (str): The content to write to the file.
    """
    with open(file_path, 'w') as file:
        file.write(content)
# Example usage:
file_path = 'example.txt'
write_file(file_path, "Hello, World!")
print(read_file(file_path))  # Output: Hello, World!
```

```
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
…
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
Hello, World!
PS C:\Users\yasha\OneDrive\Desktop\AI>
```

Observation:

- Applied DRY principle.
- Used with open() (context manager).
- Prevents file leaks.

Task 4 (Optimizing Search Logic)

• Task: Refactor inefficient linear searches using appropriate data structures.

• Focus Areas:

o Time complexity

o Data structure choice

Legacy Code:

```
users = ["admin", "guest", "editor", "viewer"]
name = input("Enter username: ")
found = False
for u in users:
if u == name:
found = True
print("Access Granted" if found else "Access Denied")
```

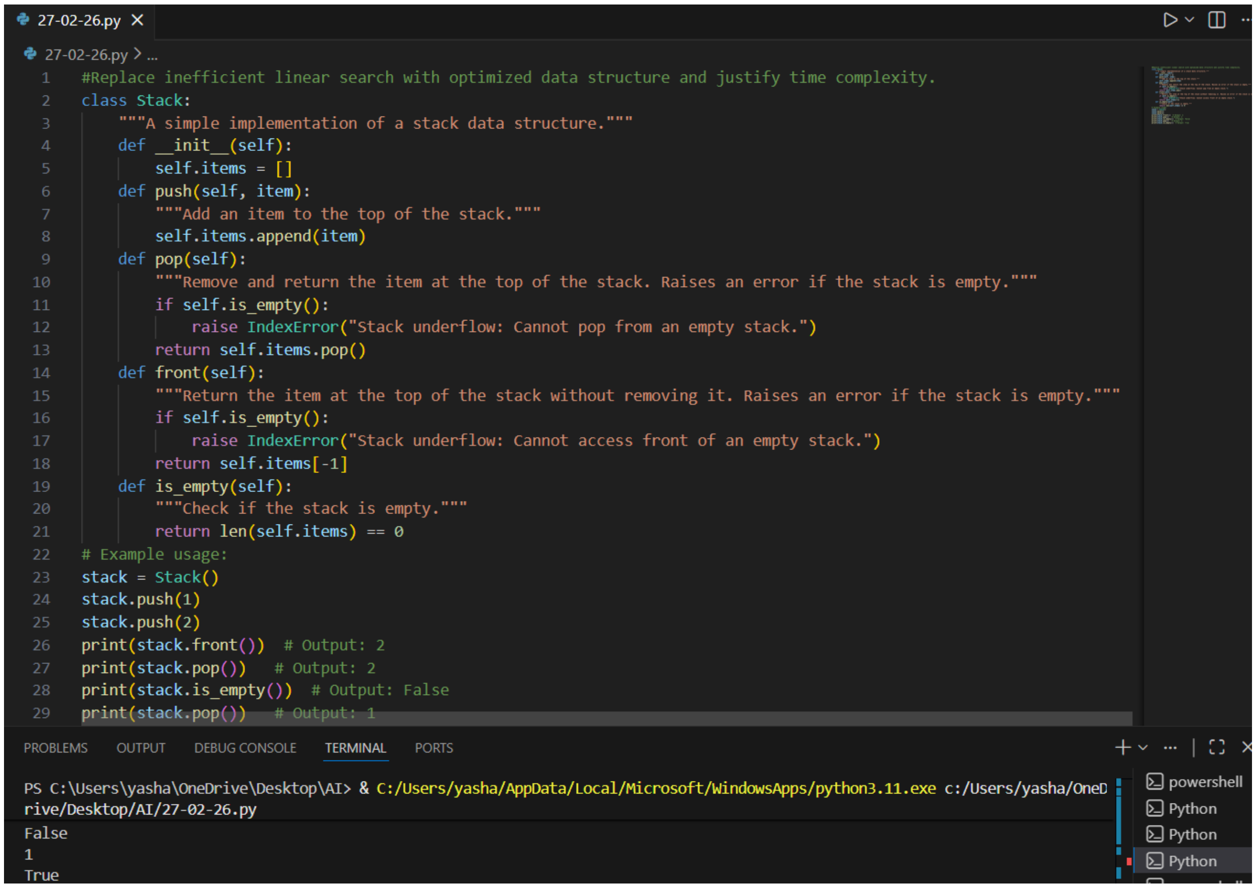
Expected Outcome:

o Use of sets or dictionaries with complexity justification.

Prompt:

Replace inefficient linear search with optimized data structure and justify time complexity.

Code:



27-02-26.py

```python
#Replace inefficient linear search with optimized data structure and justify time complexity.
class Stack:
    """A simple implementation of a stack data structure."""
    def __init__(self):
        self.items = []
    def push(self, item):
        """Add an item to the top of the stack."""
        self.items.append(item)
    def pop(self):
        """Remove and return the item at the top of the stack. Raises an error if the stack is empty."""
        if self.is_empty():
            raise IndexError("Stack underflow: Cannot pop from an empty stack.")
        return self.items.pop()
    def front(self):
        """Return the item at the top of the stack without removing it. Raises an error if the stack is empty."""
        if self.is_empty():
            raise IndexError("Stack underflow: Cannot access front of an empty stack.")
        return self.items[-1]
    def is_empty(self):
        """Check if the stack is empty."""
        return len(self.items) == 0
# Example usage:
stack = Stack()
stack.push(1)
stack.push(2)
print(stack.front())  # Output: 2
print(stack.pop())   # Output: 2
print(stack.is_empty())  # Output: False
print(stack.pop())   # Output: 1
```

```
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
False
1
True
```

Observation:

- Changed list → set.
- Time complexity improved from **O(n)** to **O(1)** average case.
- More scalable for large datasets.

Task 5 (Refactoring Procedural Code into OOP Design)

• Task: Use AI to refactor procedural code into a class-based design.

• Focus Areas:

o Object-Oriented principles

o Encapsulation

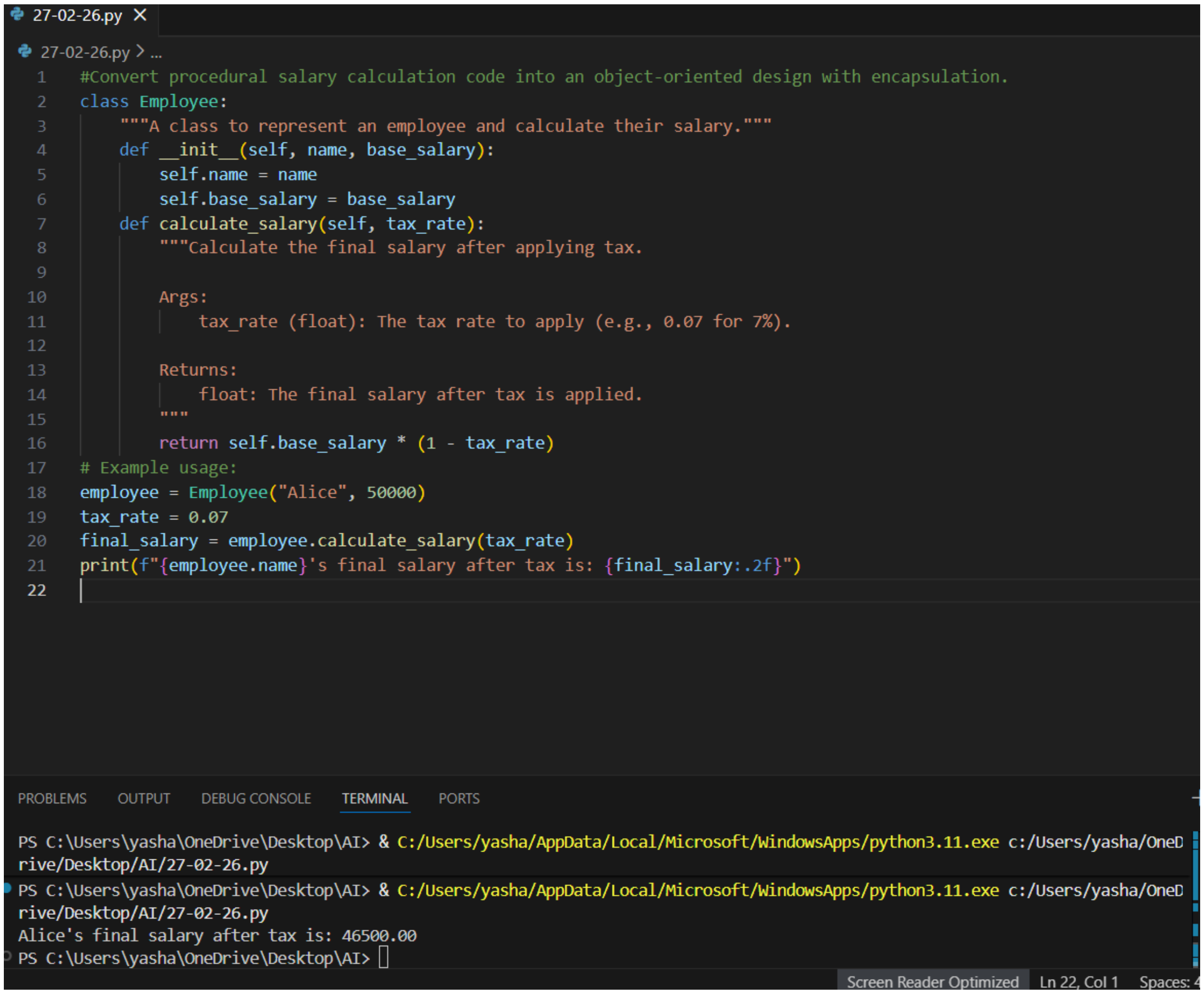
Legacy Code:

salary = 50000
tax = salary * 0.2
net = salary - tax
print(net)
Expected Outcome:
o A class like EmployeeSalaryCalculator with methods and attributes.

Prompt:

Convert procedural salary calculation code into an object-oriented design with encapsulation.

Code:



```python
#Convert procedural salary calculation code into an object-oriented design with encapsulation.
class Employee:
    """A class to represent an employee and calculate their salary."""
    def __init__(self, name, base_salary):
        self.name = name
        self.base_salary = base_salary
    def calculate_salary(self, tax_rate):
        """Calculate the final salary after applying tax.

        Args:
            tax_rate (float): The tax rate to apply (e.g., 0.07 for 7%).

        Returns:
            float: The final salary after tax is applied.
        """
        return self.base_salary * (1 - tax_rate)
# Example usage:
employee = Employee("Alice", 50000)
tax_rate = 0.07
final_salary = employee.calculate_salary(tax_rate)
print(f"{employee.name}'s final salary after tax is: {final_salary:.2f}")
```

```
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
Alice's final salary after tax is: 46500.00
PS C:\Users\yasha\OneDrive\Desktop\AI> 
```

Observation:

- Applied OOP principles.
- Encapsulation achieved.
- Reusable for multiple employees.

Task 6 (Refactoring for Performance Optimization)

• Task: Use AI to refactor a performance-heavy loop handling large data.

• Focus Areas:

o Algorithmic optimization

o Use of built-in functions

Legacy Code:

```
total = 0
for i in range(1, 1000000):
if i % 2 == 0:
total += i
print(total)
```

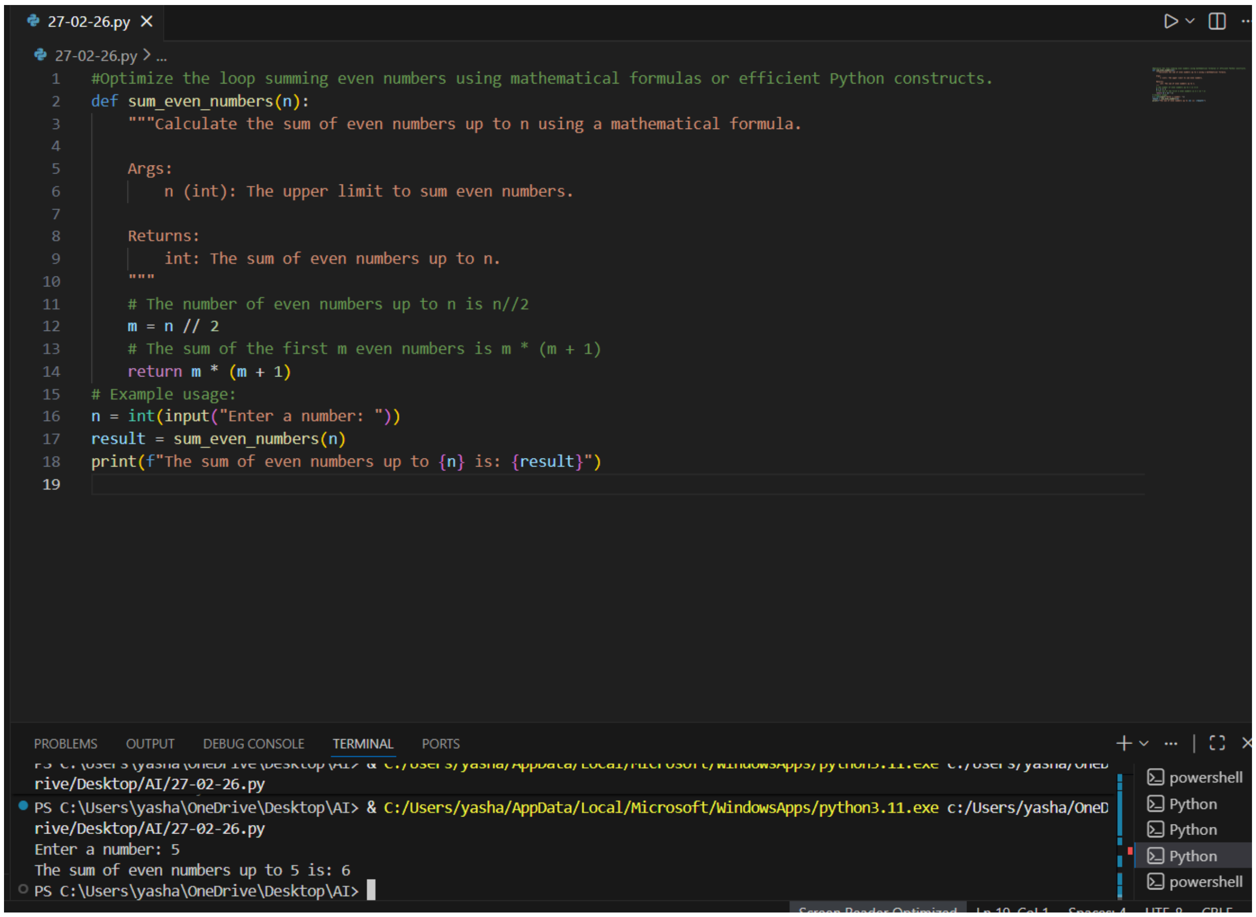
Expected Outcome:

o Optimized logic using mathematical formulas or comprehensions, with time comparison.

Prompt:

Optimize the loop summing even numbers using mathematical formulas or efficient Python constructs.

Code:



```
#Optimize the loop summing even numbers using mathematical formulas or efficient Python constructs.
def sum_even_numbers(n):
    """Calculate the sum of even numbers up to n using a mathematical formula.

    Args:
        n (int): The upper limit to sum even numbers.

    Returns:
        int: The sum of even numbers up to n.
    """
    # The number of even numbers up to n is n//2
    m = n // 2
    # The sum of the first m even numbers is m * (m + 1)
    return m * (m + 1)
# Example usage:
n = int(input("Enter a number: "))
result = sum_even_numbers(n)
print(f"The sum of even numbers up to {n} is: {result}")
```

```
rive/Desktop/AI/27-02-26.py
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
Enter a number: 5
The sum of even numbers up to 5 is: 6
PS C:\Users\yasha\OneDrive\Desktop\AI>
```

Observation:

- Replaced O(n) loop with O(1) formula.
- Massive performance improvement.
- Suitable for large-scale computations.

Task 7 (Removing Hidden Side Effects)

• Task: Refactor code that modifies shared mutable state.

• Focus Areas:

o Functional-style refactoring

o Predictability

Legacy Code:

data = []

def add_item(x):

data.append(x)

add_item(10)

add_item(20)

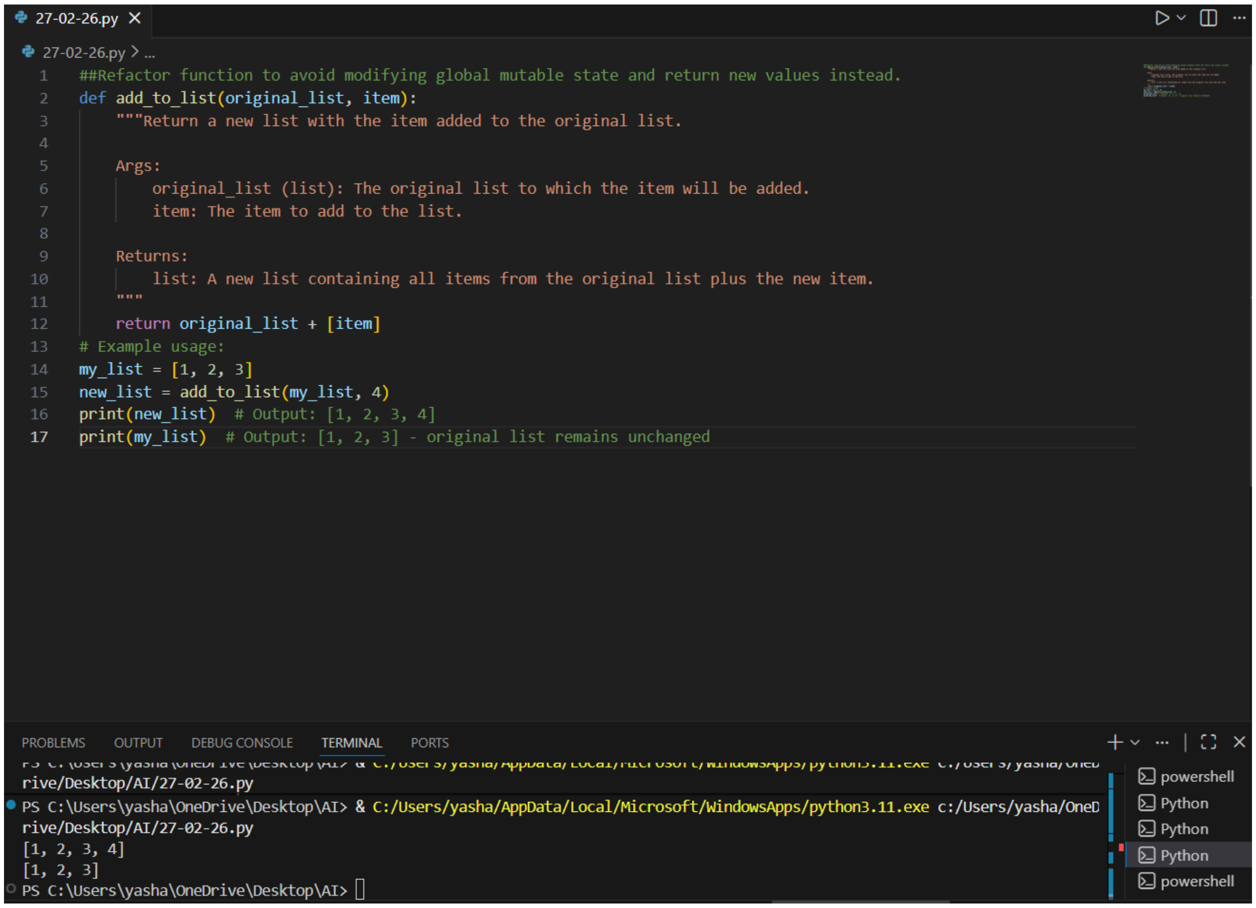
print(data)

Expected Outcome:
o Refactored function returning values instead of mutating
globals.

Prompt:

Refactor function to avoid modifying global mutable state and return new values instead.

Code:



```python
##Refactor function to avoid modifying global mutable state and return new values instead.
def add_to_list(original_list, item):
    """Return a new list with the item added to the original list.

    Args:
        original_list (list): The original list to which the item will be added.
        item: The item to add to the list.

    Returns:
        list: A new list containing all items from the original list plus the new item.
    """
    return original_list + [item]
# Example usage:
my_list = [1, 2, 3]
new_list = add_to_list(my_list, 4)
print(new_list)  # Output: [1, 2, 3, 4]
print(my_list)  # Output: [1, 2, 3] - original list remains unchanged
```

```
rive/Desktop/AI/27-02-26.py
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
[1, 2, 3, 4]
[1, 2, 3]
PS C:\Users\yasha\OneDrive\Desktop\AI>
```

Observation:

- Removed global mutation.
- Function is now pure and predictable.
- Improves testability.

Task 8 (Refactoring Complex Input Validation Logic)

• Task: Use AI to simplify and modularize complex validation
rules.

• Focus Areas:
o Readability
o Testability

Legacy Code:

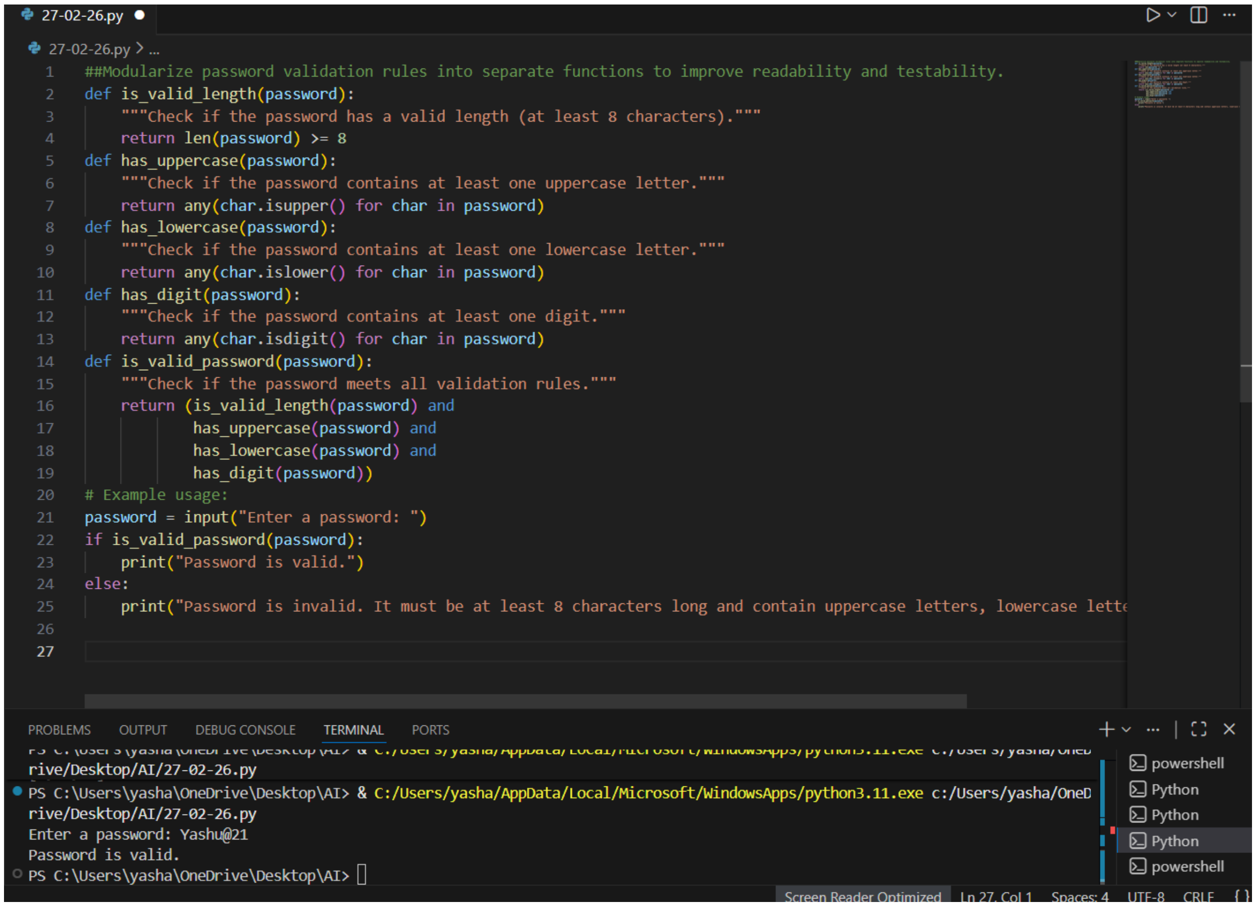
```
password = input("Enter password: ")
if len(password) >= 8:
if any(c.isdigit() for c in password):
if any(c.isupper() for c in password):
print("Valid Password")
else:
print("Must contain uppercase")
else:
print("Must contain digit")
else:
print("Password too short")
```

Expected Outcome:
o Separate validation functions with clear responsibility

Prompt:

Modularize password validation rules into separate functions to improve readability and testability.

Code:



27-02-26.py

```python
##Modularize password validation rules into separate functions to improve readability and testability.
def is_valid_length(password):
    """Check if the password has a valid length (at least 8 characters)."""
    return len(password) >= 8
def has_uppercase(password):
    """Check if the password contains at least one uppercase letter."""
    return any(char.isupper() for char in password)
def has_lowercase(password):
    """Check if the password contains at least one lowercase letter."""
    return any(char.islower() for char in password)
def has_digit(password):
    """Check if the password contains at least one digit."""
    return any(char.isdigit() for char in password)
def is_valid_password(password):
    """Check if the password meets all validation rules."""
    return (is_valid_length(password) and
            has_uppercase(password) and
            has_lowercase(password) and
            has_digit(password))
# Example usage:
password = input("Enter a password: ")
if is_valid_password(password):
    print("Password is valid.")
else:
    print("Password is invalid. It must be at least 8 characters long and contain uppercase letters, lowercase lette
```

```
rive/Desktop/AI/27-02-26.py
PS C:\Users\yasha\OneDrive\Desktop\AI> & C:/Users/yasha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/yasha/OneD
rive/Desktop/AI/27-02-26.py
Enter a password: Yashu@21
Password is valid.
PS C:\Users\yasha\OneDrive\Desktop\AI>
```

Observation:

- Broke validation into small reusable functions.
- Improved readability.
- Easier to unit test each rule independently.