AI Assisted Coding – Lab 13.5

Code Refactoring – Improving Legacy Code with AI

Student s Course Details

Field	Information
Course Code	23CS002PC304
Course Title	AI Assisted Coding
Student Name	Ravula seetharam reddy
Roll Number	2303A52440
Batch	34
Program	B.Tech – CSE (AI C ML)
Year/Sem	3rd Year – 2nd Semester
Regulation	R23

Lab Objectives

- Identify code smells and inefficiencies in legacy Python scripts.
- Use AI-assisted coding tools to refactor for readability and performance.
- Apply modern Python best practices while ensuring output correctness.

Q Task 1: Refactoring – Removing Global Variables

Objective

To eliminate global variables and improve modularity by passing required values as parameters.

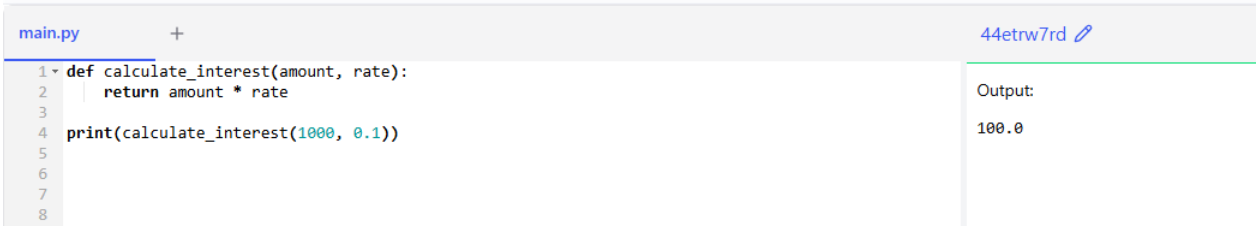
Refactored Code

```
def calculate_interest(amount, rate):
    return amount * rate

print(calculate_interest(1000, 0.1))
```

📝 Explanation

Global dependency removed. Function is now reusable, modular, and testable.

📸 Screenshots



Q Task 2: Refactoring Deeply Nested Conditionals

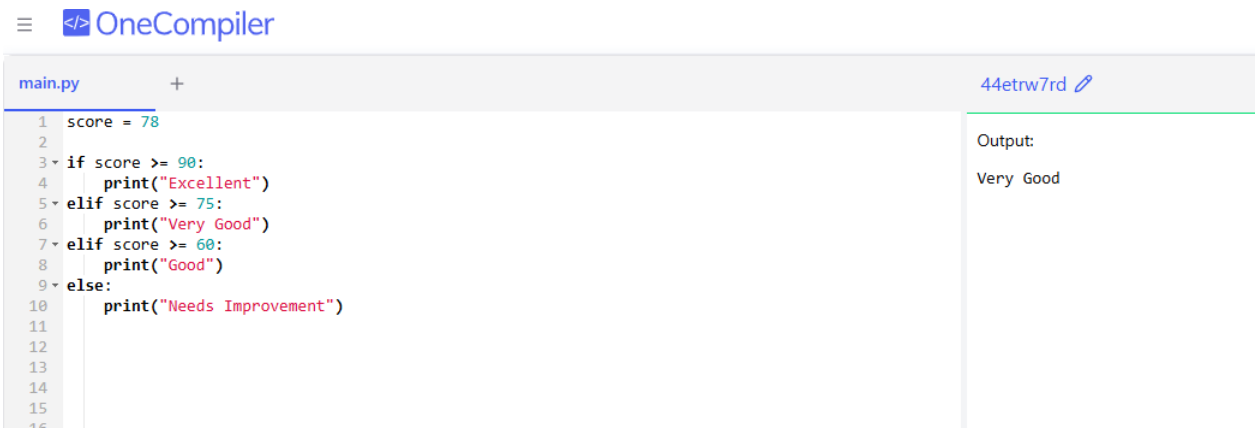
💻 Refactored Code

```
score = 78

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

📝 Explanation

Flattened nested conditions using `elif` for better readability.

Screenshots



Q Task 3: Refactoring Repeated File Handling Code

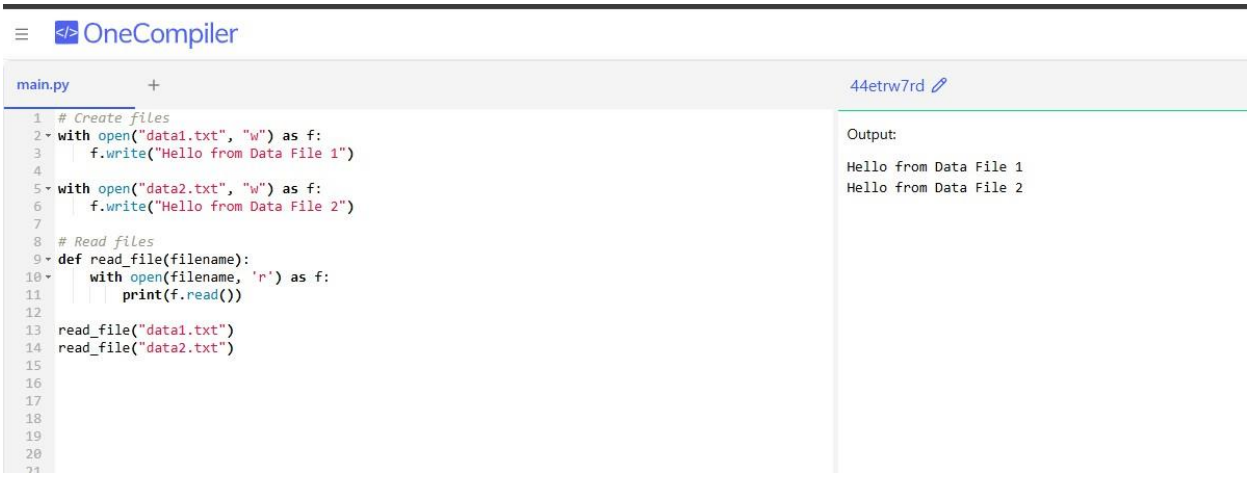
Refactored Code

```
def read_file(filename):
    with open(filename, 'r') as f:
        print(f.read())

read_file("data1.txt")
read_file("data2.txt")
```

Explanation

Used reusable function and context manager (`with`) to follow DRY principle.

Screenshots



Q Task 4: Optimizing Search Logic

Refactored Code

```
users = {"admin", "guest", "editor", "viewer"}
name = input("Enter username: ")

print("Access Granted" if name in users else "Access Denied")
```

Explanation

Used set for O(1) average lookup time instead of O(n) list search.

Screenshots

Q Task 5: Refactoring Procedural Code into OOP Design

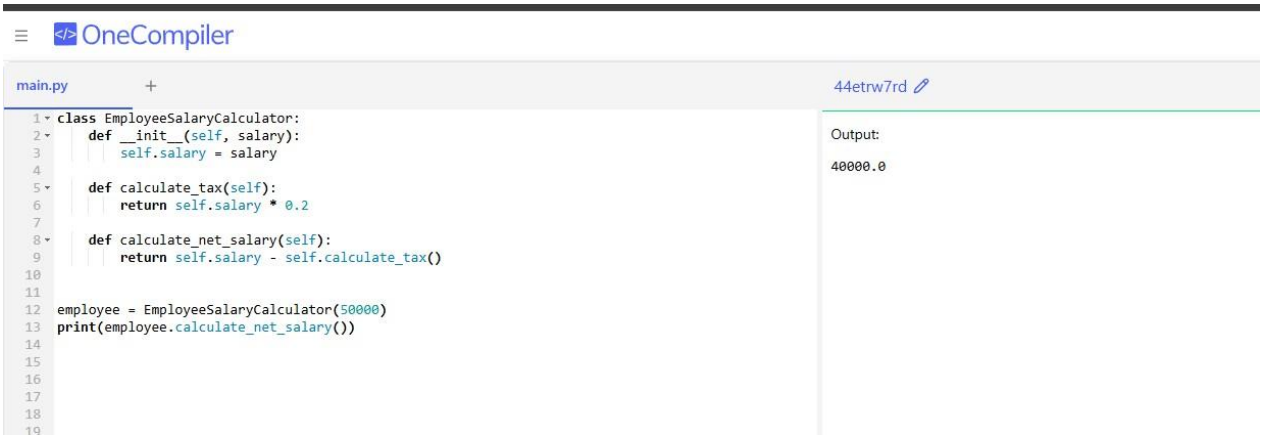
Refactored Code

```
class EmployeeSalaryCalculator:
    def __init__(self, salary):
        self.salary = salary

    def calculate_tax(self):
        return self.salary * 0.2

    def calculate_net_salary(self):
        return self.salary - self.calculate_tax()

employee = EmployeeSalaryCalculator(50000)
print(employee.calculate_net_salary())
```

Explanation

Encapsulated salary logic inside a class for better scalability and structure.

Screenshots



Q Task 6: Refactoring for Performance Optimization

Refactored Code

```
n = 1000000
count = (n - 1) // 2
last_even = count * 2
```

```
total = count * (2 + last_even) // 2
print(total)
```

Explanation

Replaced loop (O(n)) with mathematical formula (O(1)) for performance optimization.

Screenshots

Q Task 7: Removing Hidden Side Effects

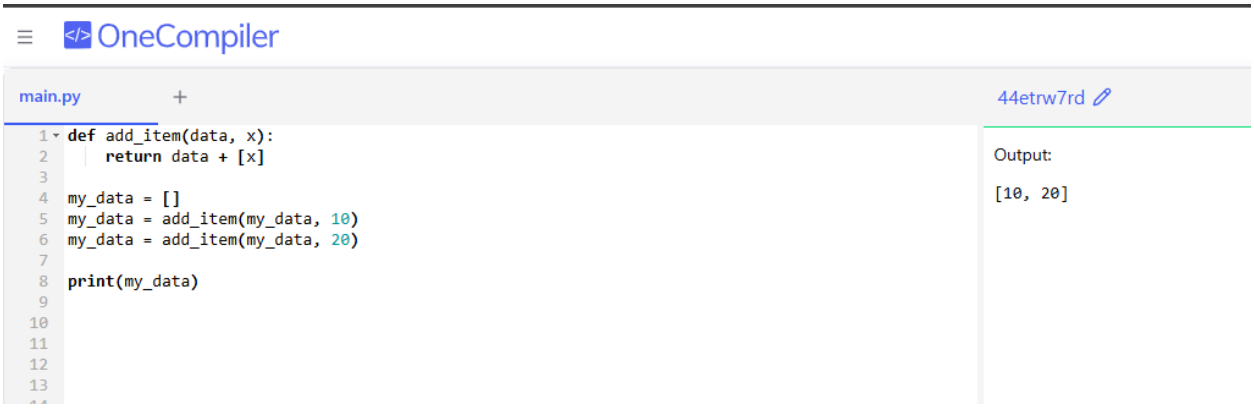
Refactored Code

```
def add_item(data, x):
    return data + [x]

my_data = []
my_data = add_item(my_data, 10)
my_data = add_item(my_data, 20)

print(my_data)
```

Explanation

Removed global mutable state. Function now returns new list instead of modifying shared data.

Screenshots



Q Task 8: Refactoring Complex Input Validation Logic

Refactored Code

```
def is_long_enough(password):
    return len(password) >= 8


def has_digit(password):
    return any(c.isdigit() for c in password)


def has_uppercase(password):
    return any(c.isupper() for c in password)


password = input("Enter password: ")

if not is_long_enough(password):
    print("Password too short")
elif not has_digit(password):
    print("Must contain digit")
elif not has_uppercase(password):
    print("Must contain uppercase")
else:
    print("Valid Password")
```

Explanation

Separated validation into small reusable functions for clarity and testability.

Screenshots

O Overall Conclusion

This lab shows how AI-assisted refactoring improves readability, maintainability, modularity, and performance. Legacy code was successfully transformed into modern, efficient, and scalable Python code using best practices.

Formatting Recommendation for Final PDF: - Font: Times New Roman - Size: 12 - Line Spacing: 1.5 - Keep headings bold - Add page breaks between tasks before final submission