Assignment-13.5

Name: Adepu Tejaswini

Ht.no: 2303A51423

Batch. No: 21

Task Description #1 (Refactoring – Removing Global Variables)

Prompt:

Refactor the given Python code to remove the global variable `rate`.

Pass the value as a function to improve modularity and testability and adding clear documentation.

Ensure the function works for different interest rates and take input from user.

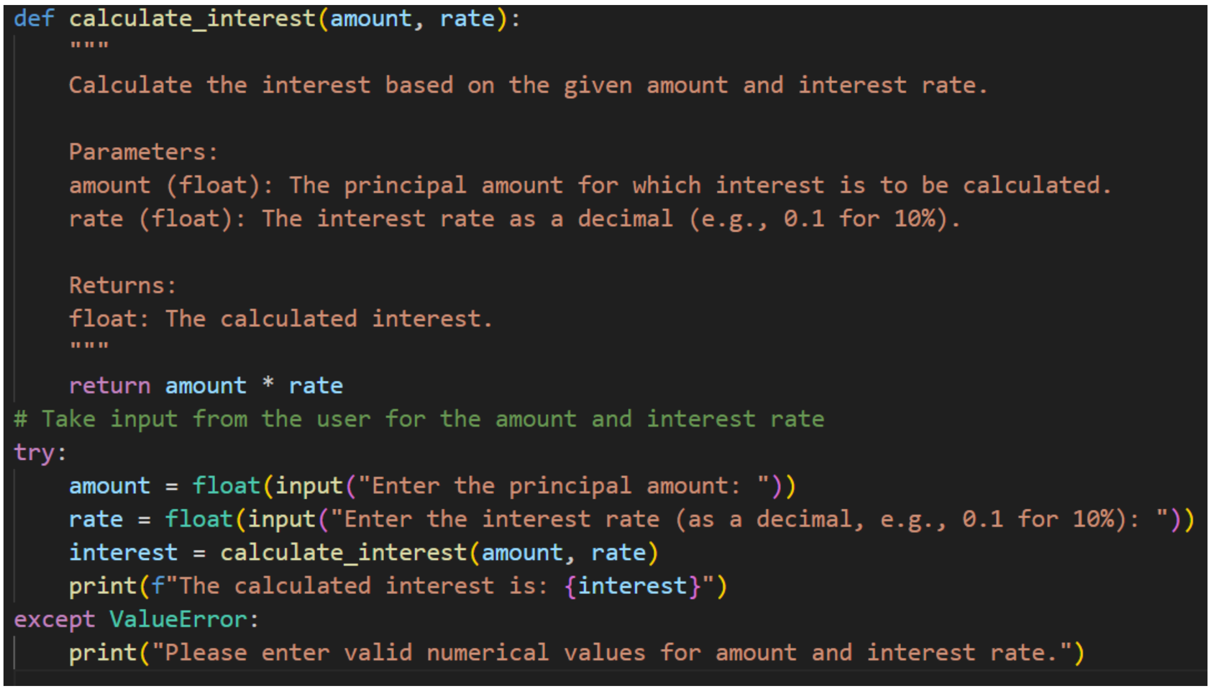
rate = 0.1

def calculate_interest(amount):

    return amount * rate

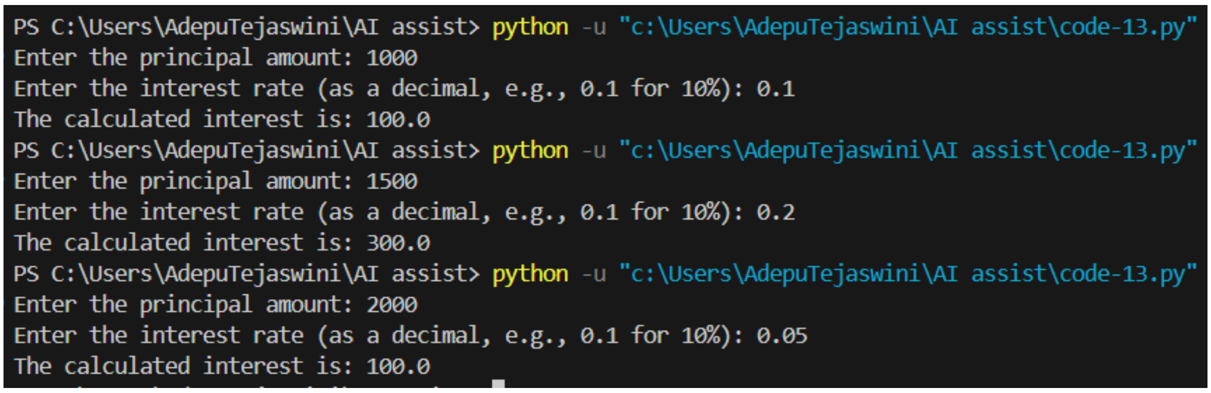
print(calculate_interest(1000))

Code:



```python
def calculate_interest(amount, rate):
    """
    Calculate the interest based on the given amount and interest rate.

    Parameters:
    amount (float): The principal amount for which interest is to be calculated.
    rate (float): The interest rate as a decimal (e.g., 0.1 for 10%).

    Returns:
    float: The calculated interest.
    """
    return amount * rate
# Take input from the user for the amount and interest rate
try:
    amount = float(input("Enter the principal amount: "))
    rate = float(input("Enter the interest rate (as a decimal, e.g., 0.1 for 10%): "))
    interest = calculate_interest(amount, rate)
    print(f"The calculated interest is: {interest}")
except ValueError:
    print("Please enter valid numerical values for amount and interest rate.")
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the principal amount: 1000
Enter the interest rate (as a decimal, e.g., 0.1 for 10%): 0.1
The calculated interest is: 100.0
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the principal amount: 1500
Enter the interest rate (as a decimal, e.g., 0.1 for 10%): 0.2
The calculated interest is: 300.0
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the principal amount: 2000
Enter the interest rate (as a decimal, e.g., 0.1 for 10%): 0.05
The calculated interest is: 100.0
```

Observation:

The updated code gets rid of the global variable and instead passes the interest rate as a parameter, making it more modular. This way, the function is easier to test with different values and doesn't rely on outside variables. The structure gets simpler, easier to reuse, and easier to keep up over time.

Task Description #2 : (Refactoring Deeply Nested Conditionals)
Prompt:

Refactor the following Python code that contains deeply nested if–elif–else statements.

Simplify the control flow using guard clauses or a mapping-based approach to improve readability and maintainability.

Remove unnecessary nesting, apply clear variable naming, and improve the logic while keeping the same output format.

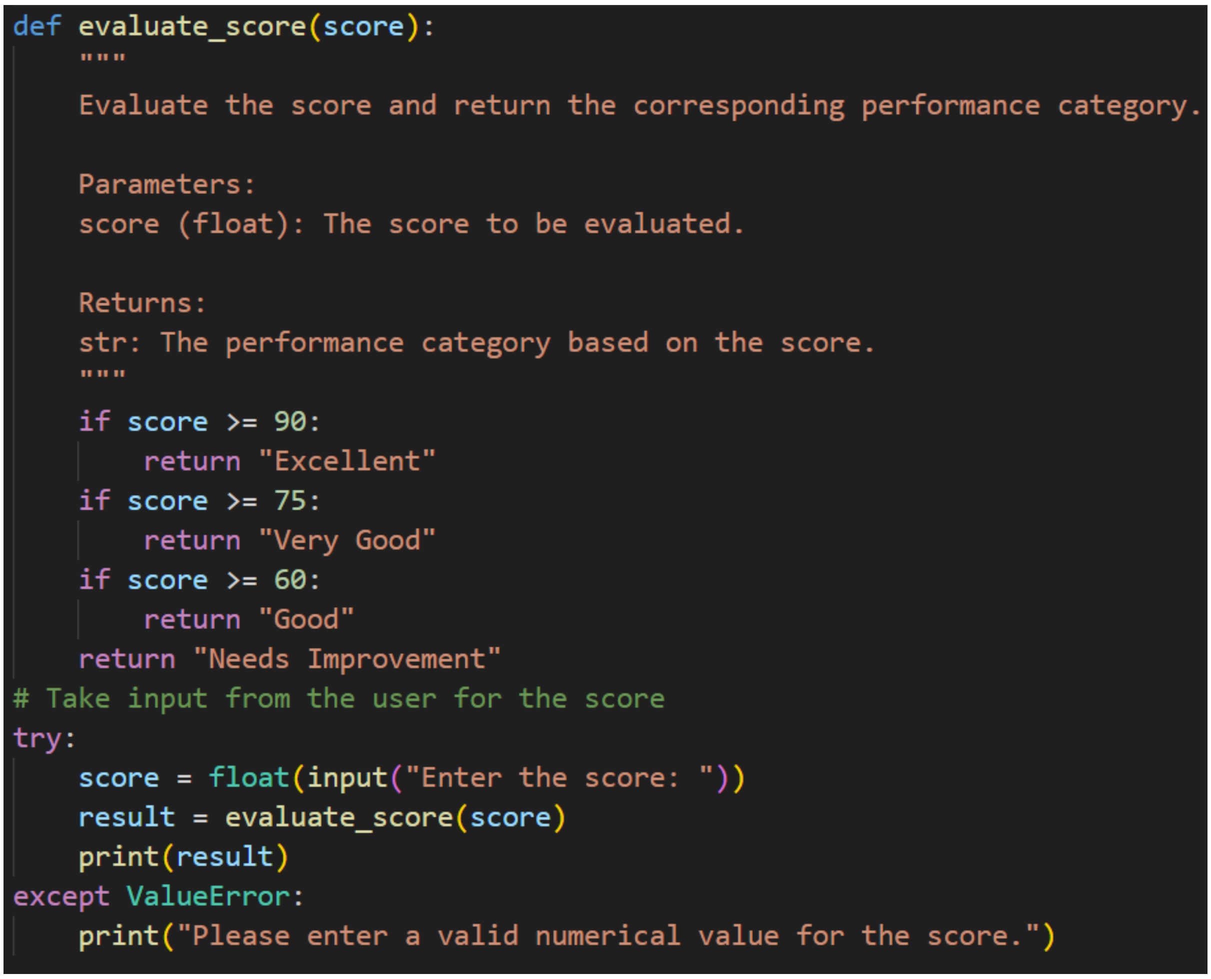
And take input from the user for the score.

Legacy Code:

```
score = 78

if score >= 90:

    print("Excellent")

else:

    if score >= 75:

        print("Very Good")

    else:
```

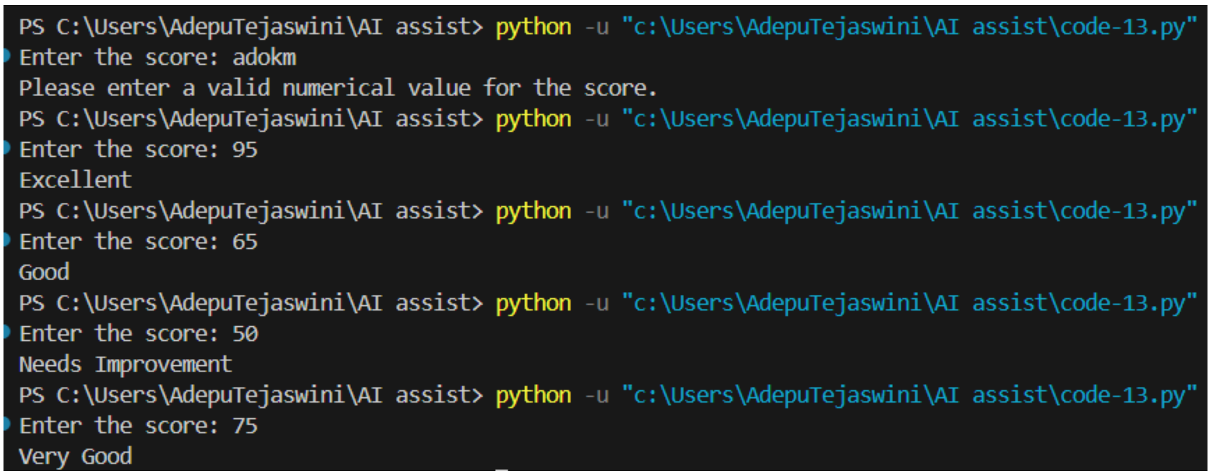
```
if score >= 60:
    print("Good")
else:
    print("Needs Improvement")
```

Code:



```python
def evaluate_score(score):
    """
    Evaluate the score and return the corresponding performance category.

    Parameters:
    score (float): The score to be evaluated.

    Returns:
    str: The performance category based on the score.
    """
    if score >= 90:
        return "Excellent"
    if score >= 75:
        return "Very Good"
    if score >= 60:
        return "Good"
    return "Needs Improvement"
# Take input from the user for the score
try:
    score = float(input("Enter the score: "))
    result = evaluate_score(score)
    print(result)
except ValueError:
    print("Please enter a valid numerical value for the score.")
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the score: adokm
Please enter a valid numerical value for the score.
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the score: 95
Excellent
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the score: 65
Good
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the score: 50
Needs Improvement
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the score: 75
Very Good
```

Observation:

The refactored code makes things easier to read by getting rid of those complicated nested ifs and using a simpler control structure instead. Simplifying the logic makes the program easier to follow and keep up, without changing what it actually does. This way, the code stays simpler and it's easier to keep up with any changes down the road.

Task Description #3 : (Refactoring Repeated File Handling Code)
Prompt:

Refactor the repeated file handling code using Python.

Apply DRY principle and context managers (with open()).

Create a reusable function that accepts the filename as a parameter, removes duplicate open/read/close code, improves variable naming,

and adds comments explaining the logic.Take input from the user for the filename.

Legacy Code:

f = open("data1.txt")

print(f.read())

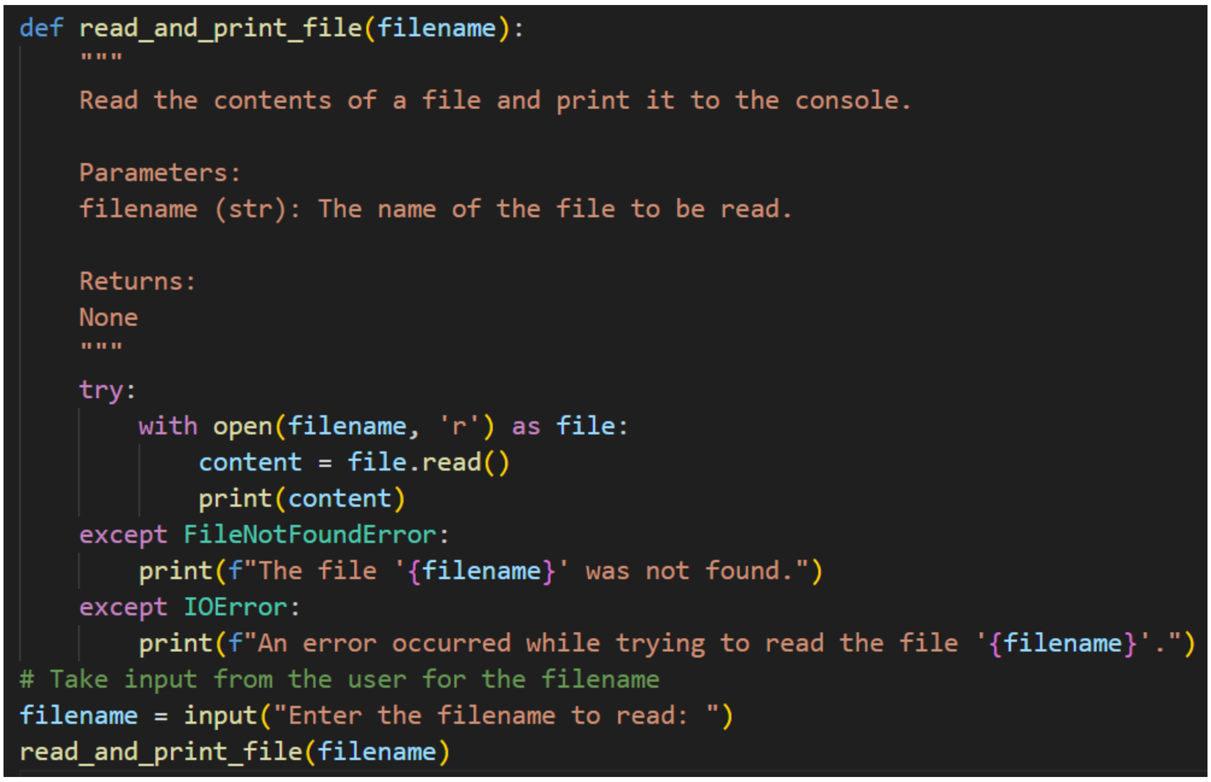
f.close()

f = open("data2.txt")

print(f.read())

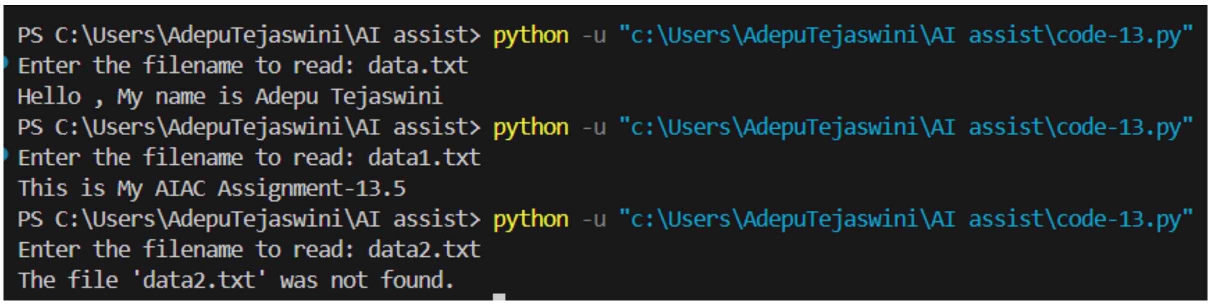
f.close()

Code:



```python
def read_and_print_file(filename):
    """
    Read the contents of a file and print it to the console.

    Parameters:
    filename (str): The name of the file to be read.

    Returns:
    None
    """
    try:
        with open(filename, 'r') as file:
            content = file.read()
            print(content)
    except FileNotFoundError:
        print(f"The file '{filename}' was not found.")
    except IOError:
        print(f"An error occurred while trying to read the file '{filename}'.")
# Take input from the user for the filename
filename = input("Enter the filename to read: ")
read_and_print_file(filename)
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the filename to read: data.txt
Hello , My name is Adepu Tejaswini
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the filename to read: data1.txt
This is My AIAC Assignment-13.5
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the filename to read: data2.txt
The file 'data2.txt' was not found.
```

Observation:

The updated code gets rid of repeating file handling parts by adding a reusable function where you just pass in the file name. Using the with open() context manager helps manage resources better by making sure files get closed automatically when you're done with them. This method sticks to the DRY principle, makes the code easier to follow, and helps keep it simple to maintain.

Task Description #4 : (Optimizing Search Logic)
Prompt:

Refactor the given Python code that performs a linear search on a list of usernames.

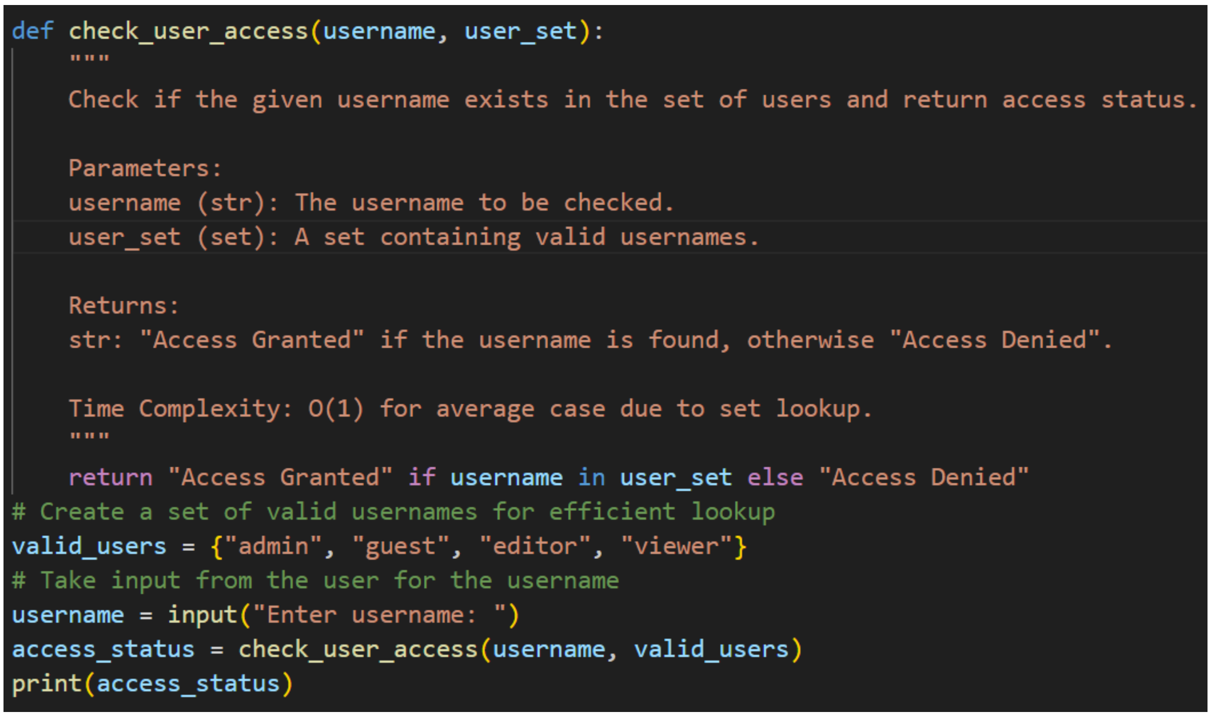
Replace the list-based linear search with a more efficient data structure like a set or dictionary to achieve better time complexity.

Include meaningful variable names, comments explaining the changes, and mention the time complexity of the new approach

and Take input from the user for the username.

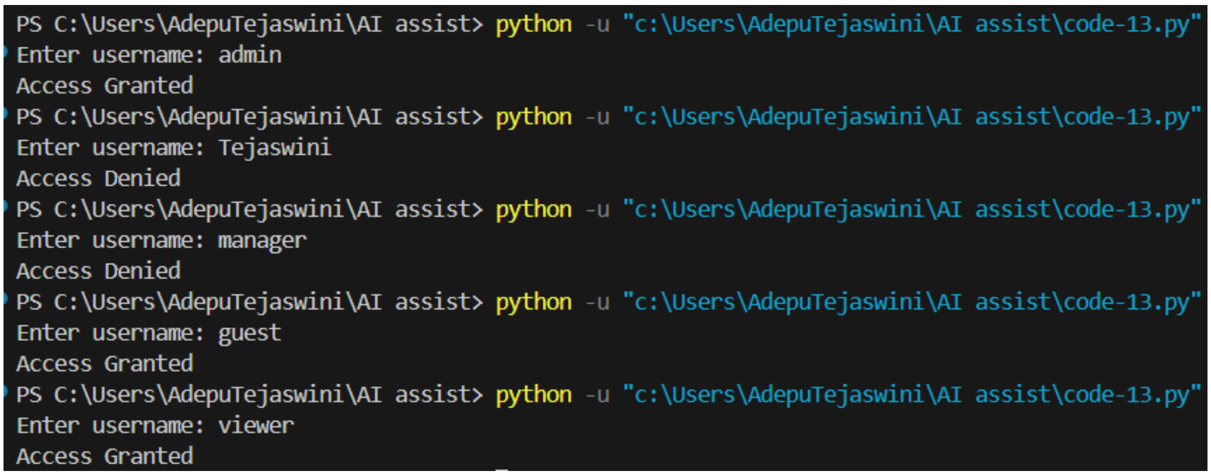
```
users = ["admin", "guest", "editor", "viewer"]

name = input("Enter username: ")

found = False

for u in users:

    if u == name:

        found = True

        break

print("Access Granted" if found else "Access Denied")
```

Code:



```python
def check_user_access(username, user_set):
    """
    Check if the given username exists in the set of users and return access status.

    Parameters:
    username (str): The username to be checked.
    user_set (set): A set containing valid usernames.

    Returns:
    str: "Access Granted" if the username is found, otherwise "Access Denied".

    Time Complexity: O(1) for average case due to set lookup.
    """
    return "Access Granted" if username in user_set else "Access Denied"
# Create a set of valid usernames for efficient lookup
valid_users = {"admin", "guest", "editor", "viewer"}
# Take input from the user for the username
username = input("Enter username: ")
access_status = check_user_access(username, valid_users)
print(access_status)
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter username: admin
Access Granted
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter username: Tejaswini
Access Denied
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter username: manager
Access Denied
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter username: guest
Access Granted
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter username: viewer
Access Granted
```

Observation:

The updated program makes searching faster by switching from a list to a set, which cuts down the lookup time from O(n) to about O(1).The control flow is easier to follow by checking membership directly instead of using a loop with a flag variable. This makes the code cleaner, easier to read, and better in performance.

Task Description #5 : (Refactoring Procedural Code into OOP Design)
Prompt:

Refactor the given Python code into an Object-Oriented design.

Create a class named EmployeeSalaryCalculator that follows OOP principles and encapsulation.

Include attributes for salary and tax_rate, and methods to calculate tax and net salary.

Adding proper documentation. Ensure the code is clean, modular, and readabile. Take input from the user for salary and tax rate.

Legacy Code:

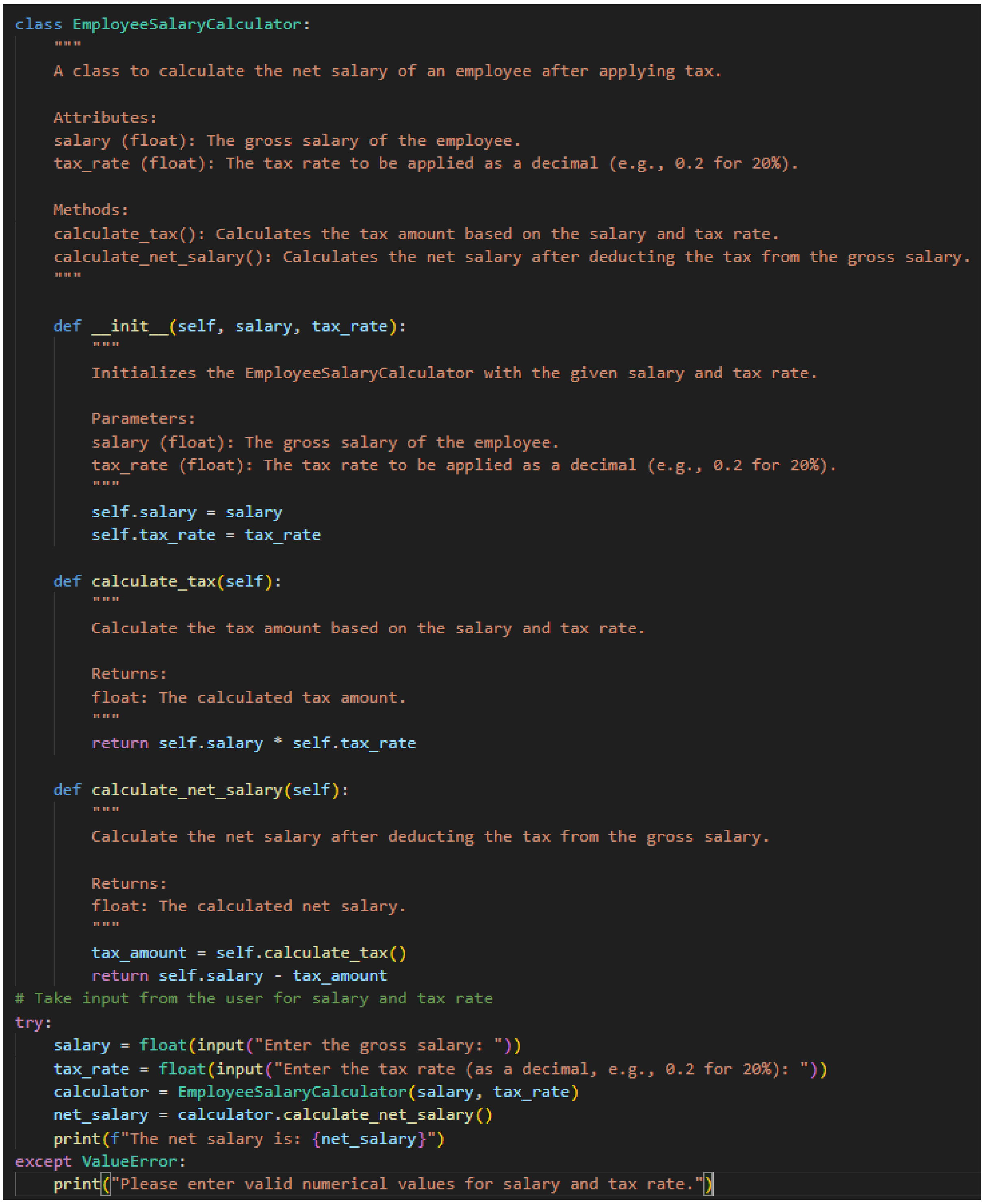
salary = 50000

tax = salary * 0.2

net = salary - tax

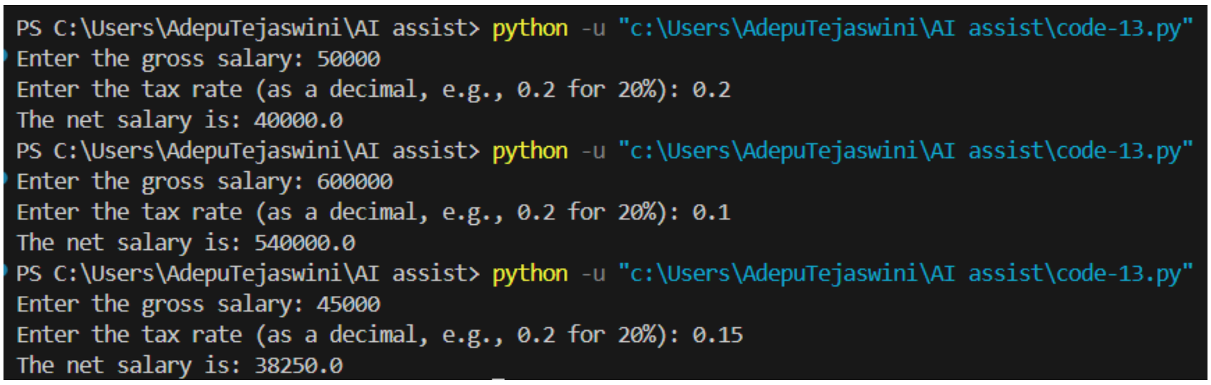
print(net)

Code:



```python
class EmployeeSalaryCalculator:
    """
    A class to calculate the net salary of an employee after applying tax.

    Attributes:
    salary (float): The gross salary of the employee.
    tax_rate (float): The tax rate to be applied as a decimal (e.g., 0.2 for 20%).

    Methods:
    calculate_tax(): Calculates the tax amount based on the salary and tax rate.
    calculate_net_salary(): Calculates the net salary after deducting the tax from the gross salary.
    """

    def __init__(self, salary, tax_rate):
        """
        Initializes the EmployeeSalaryCalculator with the given salary and tax rate.

        Parameters:
        salary (float): The gross salary of the employee.
        tax_rate (float): The tax rate to be applied as a decimal (e.g., 0.2 for 20%).
        """
        self.salary = salary
        self.tax_rate = tax_rate

    def calculate_tax(self):
        """
        Calculate the tax amount based on the salary and tax rate.

        Returns:
        float: The calculated tax amount.
        """
        return self.salary * self.tax_rate

    def calculate_net_salary(self):
        """
        Calculate the net salary after deducting the tax from the gross salary.

        Returns:
        float: The calculated net salary.
        """
        tax_amount = self.calculate_tax()
        return self.salary - tax_amount
# Take input from the user for salary and tax rate
try:
    salary = float(input("Enter the gross salary: "))
    tax_rate = float(input("Enter the tax rate (as a decimal, e.g., 0.2 for 20%): "))
    calculator = EmployeeSalaryCalculator(salary, tax_rate)
    net_salary = calculator.calculate_net_salary()
    print(f"The net salary is: {net_salary}")
except ValueError:
    print("Please enter valid numerical values for salary and tax rate.")
```

Output:


```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the gross salary: 50000
Enter the tax rate (as a decimal, e.g., 0.2 for 20%): 0.2
The net salary is: 40000.0
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the gross salary: 600000
Enter the tax rate (as a decimal, e.g., 0.2 for 20%): 0.1
The net salary is: 540000.0
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the gross salary: 45000
Enter the tax rate (as a decimal, e.g., 0.2 for 20%): 0.15
The net salary is: 38250.0
```

Observation:

The procedural code was changed into a class-based design by using object-oriented principles. Encapsulation was done by keeping salary and tax rate as attributes and using class methods to handle the calculations. The improved code is broken down into smaller parts, making it easier to reuse and simpler to keep up or add new features to.

Task Description #6 : (Refactoring for Performance Optimization)
Prompt:

Refactor the following Python code to improve performance and using built-in functions or mathematical formulas.

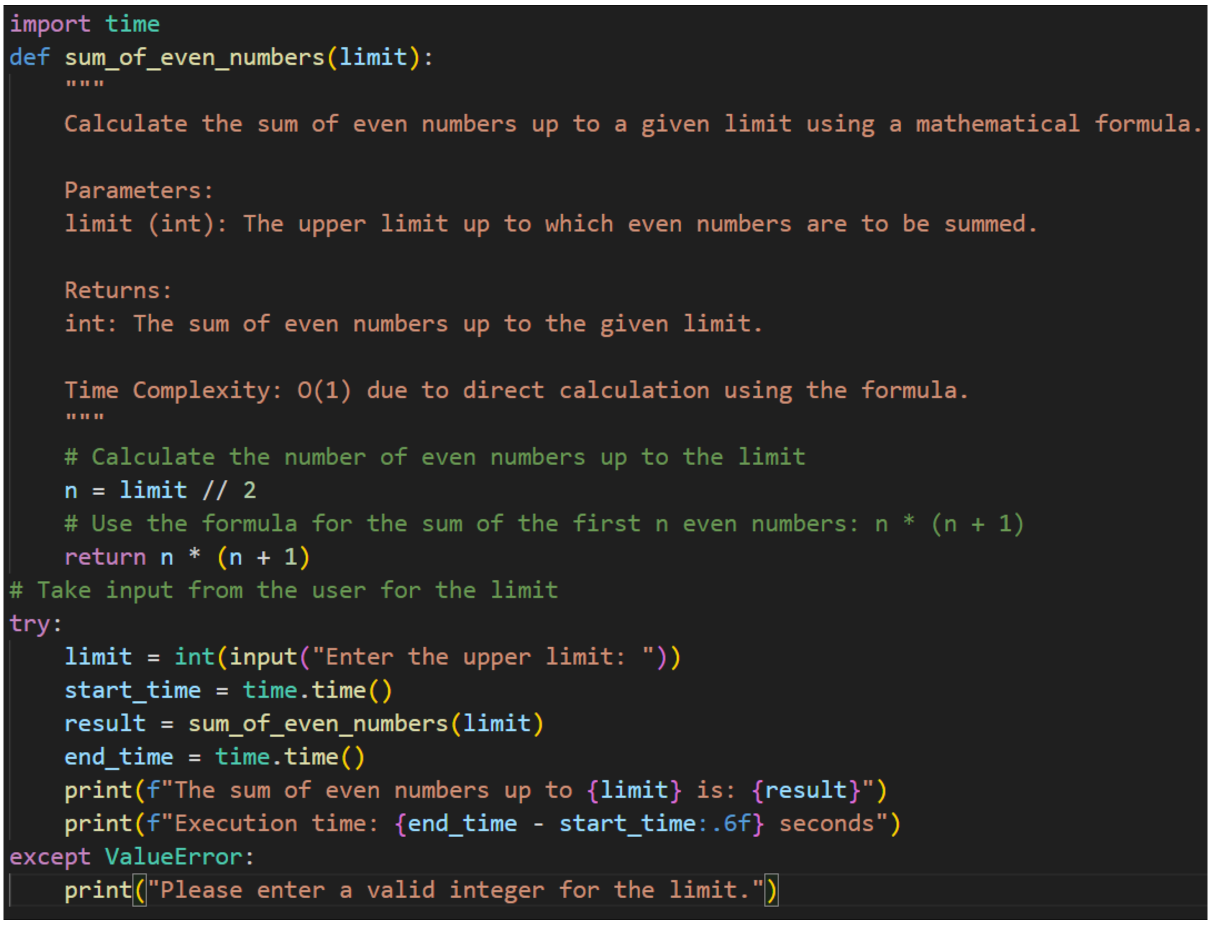
Rename variables meaningfully and improve the code structure. Compare execution time between the legacy loop implementation and the optimized version.

Take input from the user for the limit.

Legacy Code:

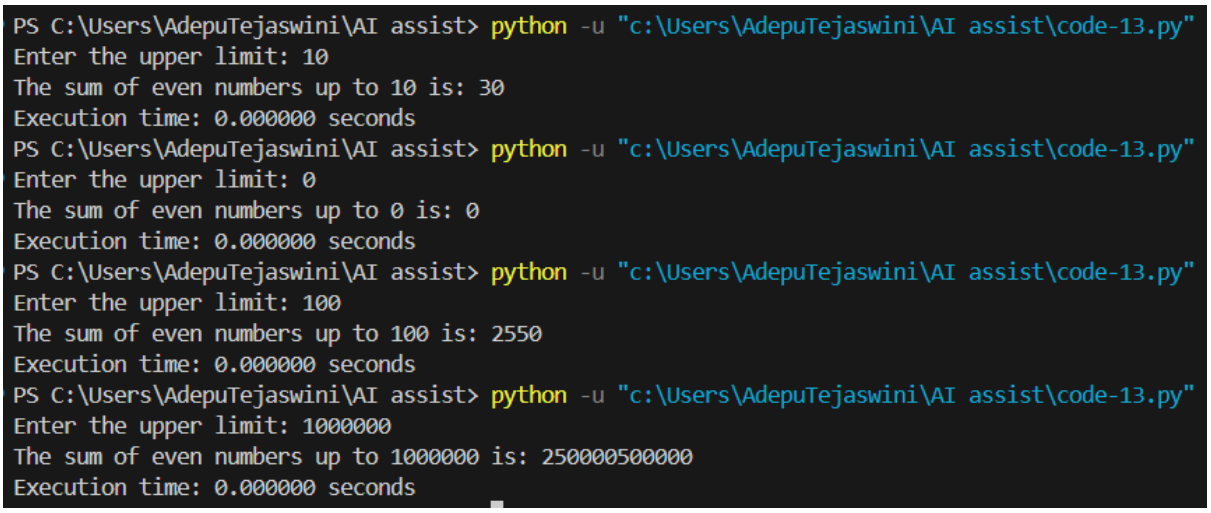
```
total = 0

for i in range(1, 1000000):

   if i % 2 == 0:

       total += i

print(total)
```

Code:



```python
import time
def sum_of_even_numbers(limit):
    """
    Calculate the sum of even numbers up to a given limit using a mathematical formula.

    Parameters:
    limit (int): The upper limit up to which even numbers are to be summed.

    Returns:
    int: The sum of even numbers up to the given limit.

    Time Complexity: O(1) due to direct calculation using the formula.
    """
    # Calculate the number of even numbers up to the limit
    n = limit // 2
    # Use the formula for the sum of the first n even numbers: n * (n + 1)
    return n * (n + 1)
# Take input from the user for the limit
try:
    limit = int(input("Enter the upper limit: "))
    start_time = time.time()
    result = sum_of_even_numbers(limit)
    end_time = time.time()
    print(f"The sum of even numbers up to {limit} is: {result}")
    print(f"Execution time: {end_time - start_time:.6f} seconds")
except ValueError:
    print("Please enter a valid integer for the limit.")
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the upper limit: 10
The sum of even numbers up to 10 is: 30
Execution time: 0.000000 seconds
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the upper limit: 0
The sum of even numbers up to 0 is: 0
Execution time: 0.000000 seconds
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the upper limit: 100
The sum of even numbers up to 100 is: 2550
Execution time: 0.000000 seconds
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter the upper limit: 1000000
The sum of even numbers up to 1000000 is: 250000500000
Execution time: 0.000000 seconds
```

Observation:

The refactored code boosts performance by swapping out the loop for a simple math formula and using built-in functions. This cuts down on the time it takes and gets rid of unneeded conditional checks inside the loop. Because of this, the improved version runs faster and gives the same results with code that's easier to read and maintain.

Task Description #7 : (Removing Hidden Side Effects)
Prompt:

Refactor the given Python code to remove hidden side effects caused by modifying a global mutable list.

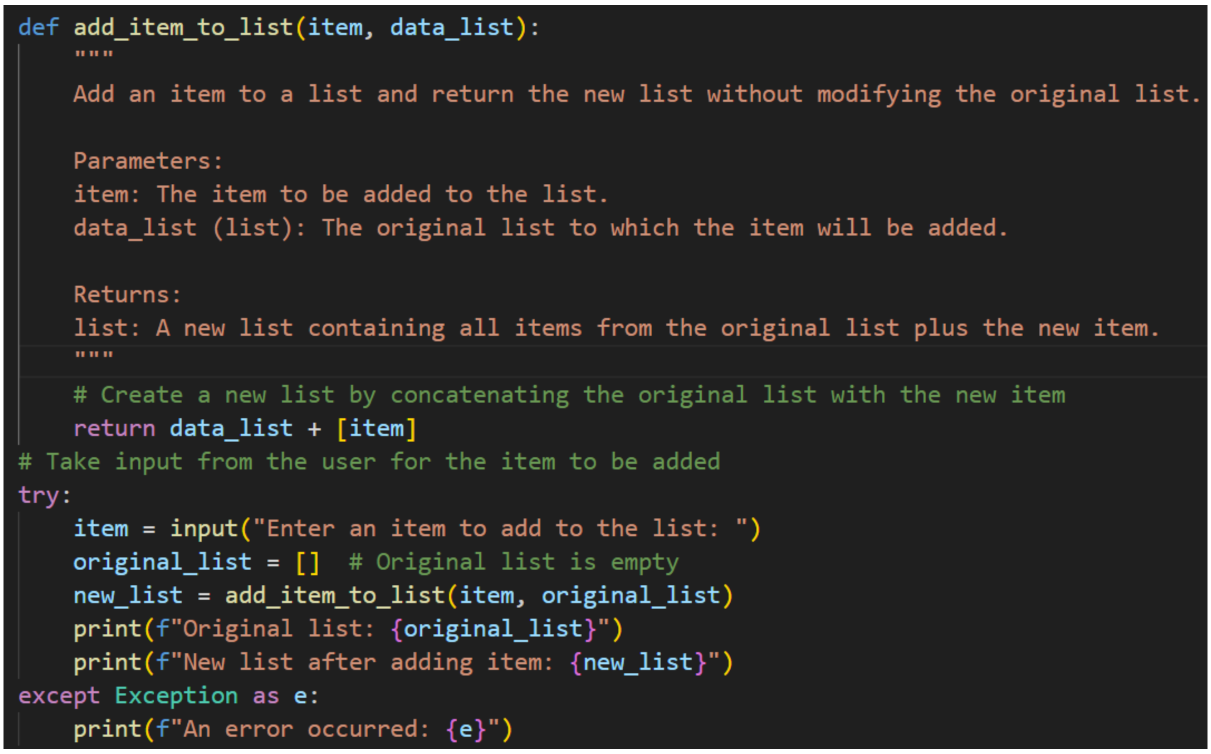
Rewrite the function in a functional style so it returns a new list instead of mutating global state.

Add clear comments and Take input from the user.

Legacy Code:

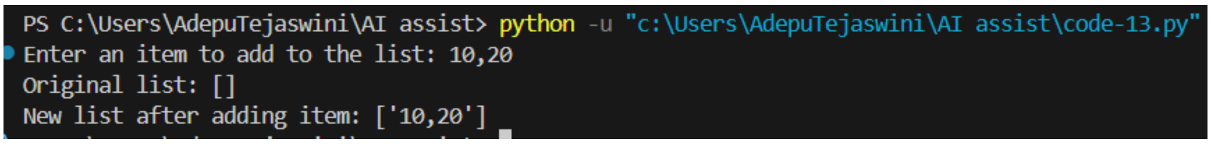
```
data = []

def add_item(x):

    data.append(x)

add_item(10)

add_item(20)

print(data)
```

Code:



```python
def add_item_to_list(item, data_list):
    """
    Add an item to a list and return the new list without modifying the original list.

    Parameters:
    item: The item to be added to the list.
    data_list (list): The original list to which the item will be added.

    Returns:
    list: A new list containing all items from the original list plus the new item.
    """
    # Create a new list by concatenating the original list with the new item
    return data_list + [item]
# Take input from the user for the item to be added
try:
    item = input("Enter an item to add to the list: ")
    original_list = []  # Original list is empty
    new_list = add_item_to_list(item, original_list)
    print(f"Original list: {original_list}")
    print(f"New list after adding item: {new_list}")
except Exception as e:
    print(f"An error occurred: {e}")
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter an item to add to the list: 10,20
Original list: []
New list after adding item: ['10,20']
```

Observation:

The rewritten code gets rid of hidden side effects by not changing the global list. The function now works in a functional programming style by creating and returning a new list instead of changing shared mutable state. This makes the code more predictable and easier to maintain, and it also simplifies testing.

Task Description #8 : (Refactoring Complex Input Validation Logic)

Prompt:

Refactor the following Python password validation code to improve readability.

Create separate functions: check_length(password), check_digit(password), and check_uppercase(password) that return True/False.

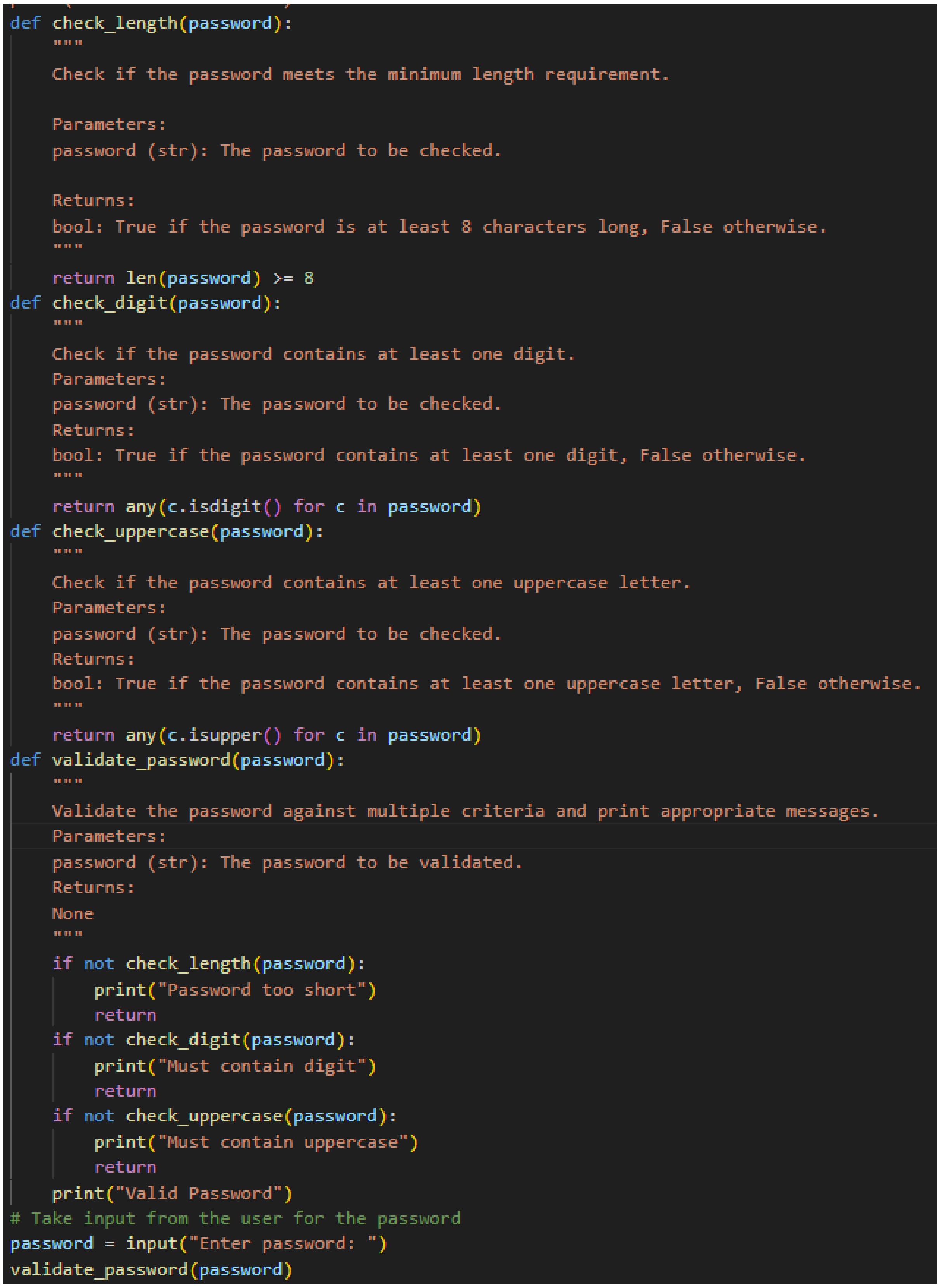
Implement a validate_password(password) function that calls these functions and prints appropriate error messages if a rule fails.

Add comments and Take input from the user.

Legacy Code:

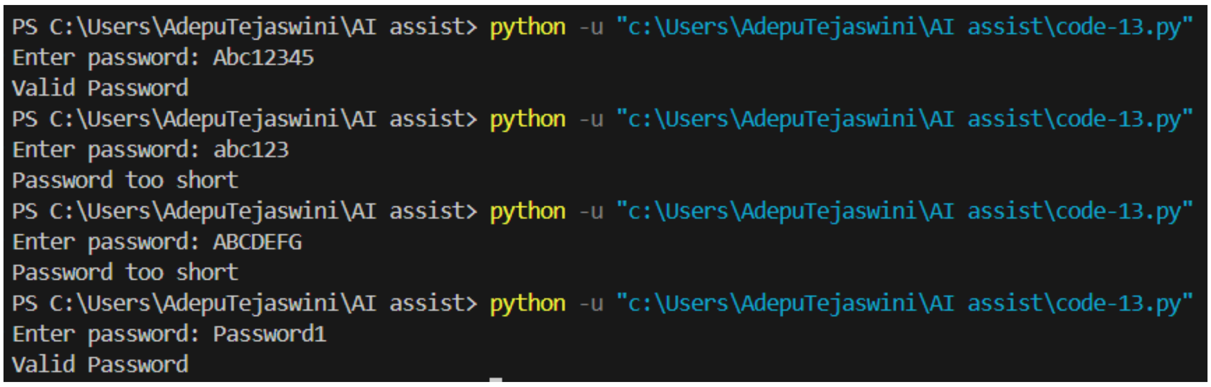
```
password = input("Enter password: ")

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")
```

Code:



```python
def check_length(password):
    """
    Check if the password meets the minimum length requirement.

    Parameters:
    password (str): The password to be checked.

    Returns:
    bool: True if the password is at least 8 characters long, False otherwise.
    """
    return len(password) >= 8
def check_digit(password):
    """
    Check if the password contains at least one digit.
    Parameters:
    password (str): The password to be checked.
    Returns:
    bool: True if the password contains at least one digit, False otherwise.
    """
    return any(c.isdigit() for c in password)
def check_uppercase(password):
    """
    Check if the password contains at least one uppercase letter.
    Parameters:
    password (str): The password to be checked.
    Returns:
    bool: True if the password contains at least one uppercase letter, False otherwise.
    """
    return any(c.isupper() for c in password)
def validate_password(password):
    """
    Validate the password against multiple criteria and print appropriate messages.
    Parameters:
    password (str): The password to be validated.
    Returns:
    None
    """
    if not check_length(password):
        print("Password too short")
        return
    if not check_digit(password):
        print("Must contain digit")
        return
    if not check_uppercase(password):
        print("Must contain uppercase")
        return
    print("Valid Password")
# Take input from the user for the password
password = input("Enter password: ")
validate_password(password)
```

Output:



```
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter password: Abc12345
Valid Password
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter password: abc123
Password too short
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter password: ABCDEFG
Password too short
PS C:\Users\AdepuTejaswini\AI assist> python -u "c:\Users\AdepuTejaswini\AI assist\code-13.py"
Enter password: Password1
Valid Password
```

Observation:

The code is easier to read. Each validation rule is in its function. This makes the program simple to test and debug and maintain. It is better, than using a lot of conditional statements. The code is easy to understand. We can reuse the validation logic. The validation logic is still the same as it was before. The code is easier to understand and reuse. The refactored code and the original code have the validation logic but the refactored code is simpler.