AI Assisted Coding

Name : Vineeth-Chidurala **Date** : 27-02-2026

Ht.No. : 2303A52447

Task - 1 : Refactoring – Removing Global Variables
• Task: Use AI to eliminate unnecessary global variables from the code.

Prompt : You are a software refactoring assistant.

Given a legacy code snippet that uses global variables, perform the following:

1. Identify all global variables used across functions.

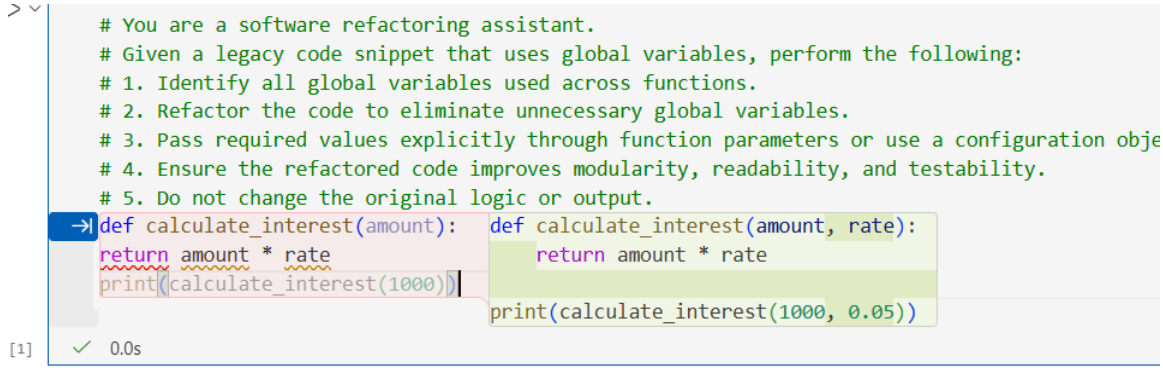
2. Refactor the code to eliminate unnecessary global variables.

3. Pass required values explicitly through function parameters or use a configuration object.

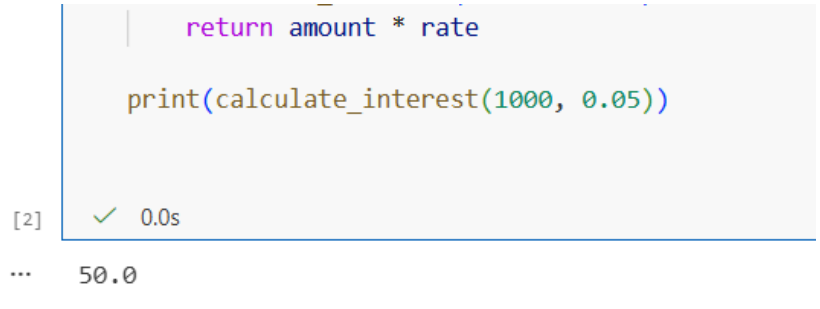
4. Ensure the refactored code improves modularity, readability, and testability.

5. Do not change the original logic or output.

Screenshots :



Output :



Task – 2 : Refactoring Deeply Nested Conditionals)

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

Prompt : You are an AI assistant specializing in code refactoring and clean code practices.

Given a legacy code snippet with deeply nested if–else conditionals, perform the following:

1. Identify deeply nested conditional logic.
2. Refactor the code to improve readability and maintainability.
3. Simplify the logic by flattening conditionals using guard clauses or a mapping-based approach.
4. Ensure the program behavior and output remain unchanged.
5. Follow clean coding principles.

ScreenShots :

```
# You are an AI assistant specializing in code refactoring and clean code practices.
# Given a legacy code snippet with deeply nested if-else conditionals, perform the following:
# 1. Identify deeply nested conditional logic.
# 2. Refactor the code to improve readability and maintainability.
# 3. Simplify the logic by flattening conditionals using guard clauses or a mapping-based approach.
# 4. Ensure the program behavior and output remain unchanged.
# 5. Follow clean coding principles.

score = 78
if score >= 90:
    print("Excellent")
else:
    if score >= 75:
```

Output :

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

[3] ✓ 0.0s

```
Very Good
```

Task-3 : Refactoring Repeated File Handling Code)

• Task: Use AI to refactor repeated file open/read/close logic.

Prompt : You are an AI assistant focused on refactoring code using clean code principles.

Given a legacy code snippet that contains repeated file open, read, and close operations, perform the following:

1. Identify repeated file handling logic.
2. Refactor the code to follow the DRY (Don't Repeat Yourself) principle.
3. Use context managers (with open()) for safe and efficient file handling.
4. Encapsulate reusable logic inside a function with parameters.
5. Ensure the original behavior and output remain unchanged.
6. Improve readability, maintainability, and testability.

ScreenShots :

```
# 4. Encapsulate reusable logic inside a function with parameters.
# 5. Ensure the original behavior and output remain unchanged.
# 6. Improve readability, maintainability, and testability.

f = open("data1.txt")
print(f.read())
f.close()

f = open("data2.txt")
print(f.read())
f.close()
```

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

read_file("data1.txt")
```

Output :

```
import os

def read_file(filename):
    if not os.path.exists(filename):
        return f"File not found: {filename}"
    with open(filename, "r") as f:
        return f.read()

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

```
0.0s
File not found: data1.txt
File not found: data2.txt
```

Task-4 : Optimizing Search Logic

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

Prompt : You are an AI assistant specializing in code optimization and data structures.

Given a legacy code snippet that performs an inefficient linear search, do the following:

1. Identify the inefficiency in the search logic.
2. Refactor the code using an appropriate data structure to improve time complexity.
3. Replace linear search with constant-time lookup where possible.
4. Ensure the program behavior and output remain unchanged.
5. Justify the choice of data structure using time complexity analysis.

Screenshots :

```
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")
```

```
found = name in users
```

Output :

```
# 5. Justify the choice of data structure using time complexity analysis.

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

```
[9]  ✓ 10.1s
...  Access Denied
```

Task-5 : Refactoring Procedural Code into OOP Design)

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

Prompt : You are an AI assistant specializing in object-oriented design and code refactoring.

Given a procedural code snippet, perform the following:

1. Identify logic that can be encapsulated into a class.
2. Refactor the code into a class-based, object-oriented design.
3. Apply object-oriented principles such as encapsulation and abstraction.
4. Use appropriate attributes and methods.
5. Ensure the original behavior and output remain unchanged.
6. Improve maintainability and reusability.

Screenshots :

```
# You are an AI assistant specializing in object-oriented design and code refactoring.
# Given a procedural code snippet, perform the following:
# 1. Identify logic that can be encapsulated into a class.
# 2. Refactor the code into a class-based, object-oriented design.
# 3. Apply object-oriented principles such as encapsulation and abstraction.
# 4. Use appropriate attributes and methods.
# 5. Ensure the original behavior and output remain unchanged.
# 6. Improve maintainability and reusability.

salary = 50000
tax = salary * 0.2
net = salary - tax
print(net)
```

Output :

```
# 4. Use appropriate attributes and methods.
# 5. Ensure the original behavior and output remain unchanged.
# 6. Improve maintainability and reusability.

salary = 50000
tax = salary * 0.2
net = salary - tax
print(net)
```

[10] ✓ 0.0s

```
40000.0
```

Task-6 : Refactoring for Performance Optimization

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

Prompt : You are an AI assistant specializing in performance optimization and efficient algorithms.

Given a legacy code snippet that uses a performance-heavy loop on large data, perform the following:

1. Identify the performance bottleneck in the code.
2. Refactor the logic to reduce time complexity.
3. Use optimized approaches such as mathematical formulas, built-in functions, or comprehensions.
4. Ensure the program output remains unchanged.
5. Compare the time complexity of the original and optimized versions.
6. Improve readability and efficiency.

Screenshots :

```
# You are an AI assistant specializing in performance optimization and efficient algorithms.
# Given a legacy code snippet that uses a performance-heavy loop on large data, perform the following:
# 1. Identify the performance bottleneck in the code.
# 2. Refactor the logic to reduce time complexity.
# 3. Use optimized approaches such as mathematical formulas, built-in functions, or comprehensions.
# 4. Ensure the program output remains unchanged.
# 5. Compare the time complexity of the original and optimized versions.
# 6. Improve readability and efficiency.

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

Output :

```
    # 3. Use optimized approaches such as mathematical formulas, built-in functions, or comprehensions.
    # 4. Ensure the program output remains unchanged.
    # 5. Compare the time complexity of the original and optimized versions.
    # 6. Improve readability and efficiency.

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

✓ 0.2s

```
249999500000
```

Task-7 : Removing Hidden Side Effects)

• Task: Refactor code that modifies shared mutable state.

Prompt :

Rewrite the following code to remove hidden side effects.

Avoid modifying global or shared mutable state.

Make the function pure by returning new values instead.

Screenshots :

```
# Rewrite the following code to remove hidden side effects.
# Avoid modifying global or shared mutable state.
# Make the function pure by returning new values instead.

data = []
def add_item(x):
    data.append(x)
add_item(10)
add_item(20)
print(data)
```

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

data = add_item(10, data)
data = add_item(20, data)
```

Output :

```
# Rewrite the following code to remove hidden side effects.
# Avoid modifying global or shared mutable state.
# Make the function pure by returning new values instead.

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

data = add_item(10, data)
data = add_item(20, data)
print(data)
```

✓ 0.0s

```
[10, 20]
```

Task-8 : Refactoring Complex Input Validation Logic)

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

Prompt :

Break the following password validation logic into small, readable functions.
Each function should validate one rule.
Improve readability and testability without changing behavior.

Screenshots :

```
# Break the following password validation logic into small, readable functions.
# Each function should validate one rule.
# Improve readability and testability without changing behavior.

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")
```

Output :

```
# Break the following password validation logic into small, readable functions.
# Each function should validate one rule.
# Improve readability and testability without changing behavior.

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")
```

✓ 4.6s

```
Valid Password
```