AI ASSISTANT CODING - ASSIGNMENT-13.4

Name: Valaboju Harshavardhanu

Roll No: 2303A52354 **Batch:** 41

Task 1: Refactoring Data Transformation Logic

Prompt:

Refactor the code using list comprehensions or helping functions preserving the same output and initialize the list with the user input

Code:

```
values = [2, 4, 6, 8, 10]
doubled = []
for v in values:
    doubled.append(v * 2)
print(doubled)

#refactor the code using list comprehensions or helping functions preserving the
values = [int(x) for x in input("Enter numbers separated by space: ").split()]
doubled = [v * 2 for v in values]
print(doubled)
```

Output:

```
[4, 8, 12, 16, 20]
Enter numbers separated by space: 2 5 6 4 8 9 7 5 4 6 4
[4, 10, 12, 8, 16, 18, 14, 10, 8, 12, 8]
PS C:\Users\harsh\OneDrive\Desktop\AI Assist lab>
```

Justification:

This task improves code readability by converting loops into list comprehensions. Pythonic code is shorter and easier to understand. AI suggestions help refactor legacy code efficiently. The output remains unchanged after refactoring. This task helps learn clean coding practices.

Task 2: Improving Text Processing Code Readability

Prompt:

Refactor the code with more efficient and readable approach while keeping the same output and initialize with user input

Code:

```
words = ["Refactoring", "with", "AI", "improves","quality"]
message = ""
for w in words:
    message += w + " "
print(message.strip())

#Refactor the code with more efficient and readable approach w
words = input("Enter words separated by space: ").split()
message = " ".join(words)
print(message)
```

Output:

```
Refactoring with AI improves quality
Enter words separated by space: The above words are joined together
The above words are joined together
PS C:\Users\harsh\OneDrive\Desktop\AI Assist lab>
```

Justification:

This task improves string handling by replacing repeated concatenation with efficient methods. AI suggestions make the code more readable and faster. The final output remains the same as the legacy code. It helps reduce unnecessary operations in programs. This task improves understanding of efficient string processing.

Task 3: Safer Access to Configuration Data

Prompt:

Refactor the code using safer dictionary access methods the behaviour remains same for missing keys

Code:

```
config = {"host": "localhost", "port": 8080}
if "timeout" in config:
    print(config["timeout"])
else:
    print("Default timeout used")
#Refactor the code using safer dictionary access methods th
config = {"host": "localhost", "port": 8080}
timeout = config.get("timeout", "Default timeout used")
print(timeout)
```

Output:

```
Default timeout used
Default timeout used
```

Justification:

This task improves code safety by using better dictionary access methods. AI suggestions help avoid errors when keys are missing. The program behaves correctly even when values are not present. It improves reliability of configuration systems. This task helps learn safer coding techniques.

Task 4: Refactoring Conditional Logic for Scalability

Prompt:

Refactor the logic using mapping techniques while preserving the same output and initialize the action and numbers with user input

Code:

```
action = "divide"
x, y = 10, 2

if action == "add":
    result = x + y
elif action == "subtract":
    result = x - y
elif action == "multiply":
    result = x * y
elif action == "divide":
    result = x / y
else:
    result = None

print(result)
#refactor the logic using mapping techniques while preserving the same outpu
action = input("Enter action (add, subtract, multiply, divide): ")
x, y = map(float, input("Enter two numbers separated by space: ").split())
operations = {
    "add": lambda a, b: a + b,
    "subtract": lambda a, b: a - b,
    "multiply": lambda a, b: a * b,
    "divide": lambda a, b: a / b
}
result = operations.get(action, lambda a, b: None)(x, y)
print(result)
```

Output:

```
5.0
Enter action (add, subtract, multiply, divide): multiply
Enter two numbers separated by space: 5 1
5.0
PS C:\Users\harsh\OneDrive\Desktop\AI Assist lab>
```

Justification:

This task improves scalability by replacing long if-elif conditions with mapping techniques. AI suggestions make the code cleaner and easier to maintain. New operations can be added easily in the future. The functionality remains unchanged after refactoring. This task improves structured programming skills.

Task 5: Simplifying Search Logic in Collections

Prompt:

Refactor the logic into a more concise and readable form while preserving the same output and initialize the inventory with user input

Code:

```
inventory = ["pen", "notebook", "eraser", "marker"]
found = False
for item in inventory:
    if item == "eraser":
        found = True
        break
print("Item Available" if found else "Item Not Available")
#Refactor the logic into a more concise and readable form while preservir
inventory = input("Enter inventory items separated by space: ").split()
found = "eraser" in inventory
print("Item Available" if found else "Item Not Available")
```

Output:

```
Item Available
Enter inventory items separated by space: pen book eraser marker table
Item Available
PS C:\Users\harsh\OneDrive\Desktop\AI Assist lab>
```

Justification:

This task simplifies searching logic by replacing manual loops with concise methods. AI suggestions improve readability and reduce code length. The output behaviour remains the same as before. It improves efficiency in searching collections. This task helps learn better Python coding practices.