Assignment-13.4

2303A53016
Batch-46
K . Jyoshno Pranav

Lab 13: Code Refactoring – Improving Legacy Code with AI Suggestions

Task 1:

Refactoring Data Transformation Logic

Scenario

You are maintaining a data preprocessing script where numerical transformations are written using verbose loops.

Task Description

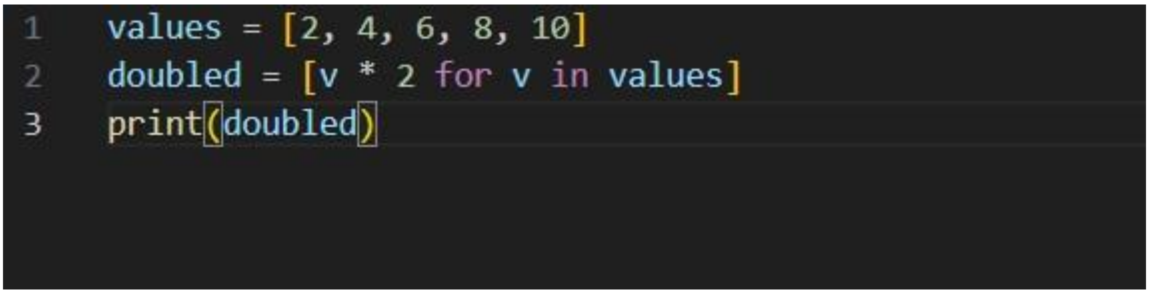
- Review the legacy code that computes transformed values using an explicit loop.
- Use an AI tool to suggest a more Pythonic refactoring approach. • Refactor the code using list comprehensions or helper functions while preserving the output. **Legacy Code**

values = [2, 4, 6, 8, 10]
doubled = [] for v in
values:
doubled.append(v * 2)
print(doubled) **Expected**
Output [4, 8, 12, 16, 20]

AI Prompt Used:

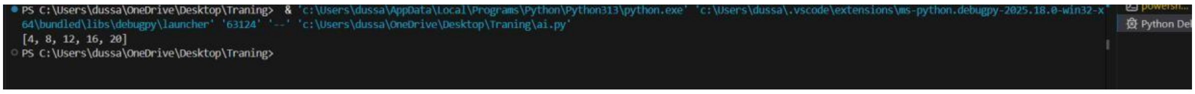
Refactor the following Python code to make it more Pythonic and concise without changing its output.
Use modern Python best practices such as list comprehensions if applicable.

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



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

Output:



```
PS C:\Users\dussa\OneDrive\Desktop\Traning>  & 'c:\Users\dussa\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\dussa\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63124' '--' 'c:\Users\dussa\OneDrive\Desktop\Traning\ai.py'
[4, 8, 12, 16, 20]
PS C:\Users\dussa\OneDrive\Desktop\Traning>
```

Task 2:

Improving Text Processing Code Readability

Scenario

You are working on a log message generator that builds messages word by word.

Task Description

- Analyze the legacy code that constructs a sentence using repeated string concatenation.
- Ask AI to suggest a more efficient and readable approach.
- Refactor the code accordingly while keeping the final output unchanged.

Legacy Code

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

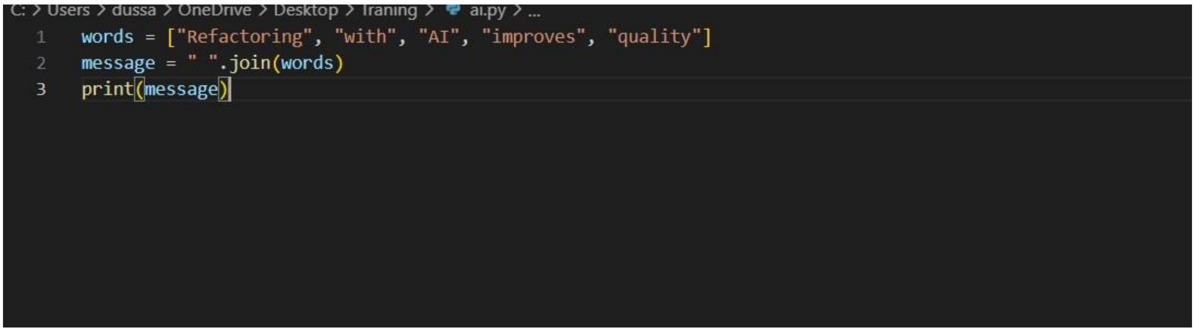
Expected Output

Refactoring with AI improves quality

AI Prompt Used:

Improve the readability and efficiency of the following Python code.
Avoid repeated string concatenation and suggest a more efficient approach while preserving the same output.

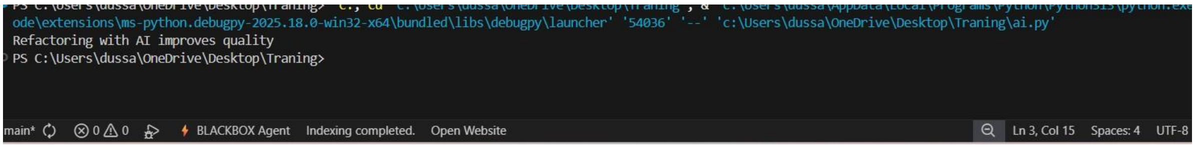
```
words = ["Refactoring", "with", "AI", "improves", "quality"]
message = "" for w in words:
  message += w + " " print(message.strip())
```



C: > Users > dussa > OneDrive > Desktop > Traning > ai.py > ...

```
words = ["Refactoring", "with", "AI", "improves", "quality"]
message = " ".join(words)
print(message)
```

Output:



```
PS C:\Users\dussa\OneDrive\Desktop\Traning> c:; cd 'c:\Users\dussa\OneDrive\Desktop\Traning'; & 'c:\Users\dussa\AppData\Local\Programs\Python\Python313\python.exe
ode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54036' '--' 'c:\Users\dussa\OneDrive\Desktop\Traning\ai.py'
Refactoring with AI improves quality
PS C:\Users\dussa\OneDrive\Desktop\Traning>
```

Task 3:

Safer Access to Configuration Data

Scenario

You are maintaining a configuration loader where dictionary keys may or may not exist depending on deployment settings.

Task Description

- Review the legacy code that manually checks for dictionary keys. • Use AI suggestions to refactor the code using safer dictionary access methods.
- Ensure the behavior remains the same for missing keys

Legacy Code

config = {"host": "localhost", "port": 8080}
if "timeout" in config:
print(config["timeout"]) else:
print("Default timeout used")

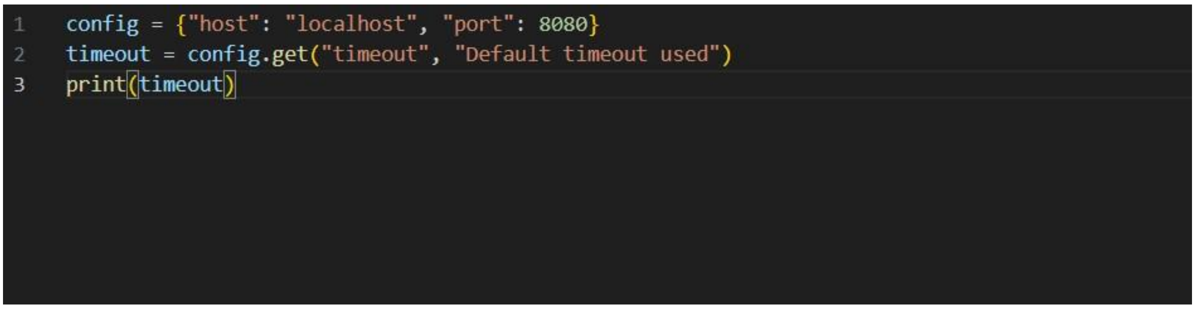
Expected Output

Default timeout used

AI Prompt Used:

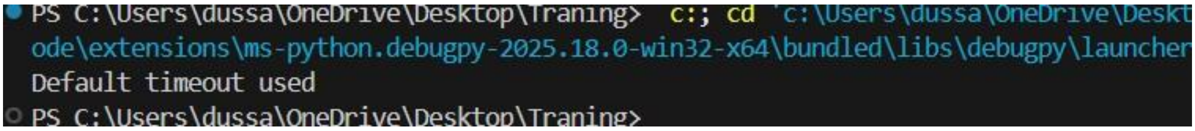
Refactor the following Python code to safely access dictionary values using modern Python methods.
Ensure that the behavior remains the same if the key does not exist.

config = {"host": "localhost", "port": 8080} if
"timeout" in config:
 print(config["timeout"]) else:
print("Default timeout used")



```
config = {"host": "localhost", "port": 8080}
timeout = config.get("timeout", "Default timeout used")
print(timeout)
```

Output:



```
PS C:\Users\dussa\OneDrive\Desktop\Traning>  c:; cd 'c:\Users\dussa\OneDrive\Deskt
ode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher
Default timeout used
PS C:\Users\dussa\OneDrive\Desktop\Traning>
```

Task 4:

Refactoring Conditional Logic for Scalability

Scenario

You are enhancing a utility module that performs different operations based on user input.

Task Description

- Examine the multiple if–elif conditions used to determine operations.
- Ask AI to suggest a cleaner, scalable alternative.

- Refactor the logic using mapping techniques while preserving functionality. **Legacy 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)
```

Expected Output

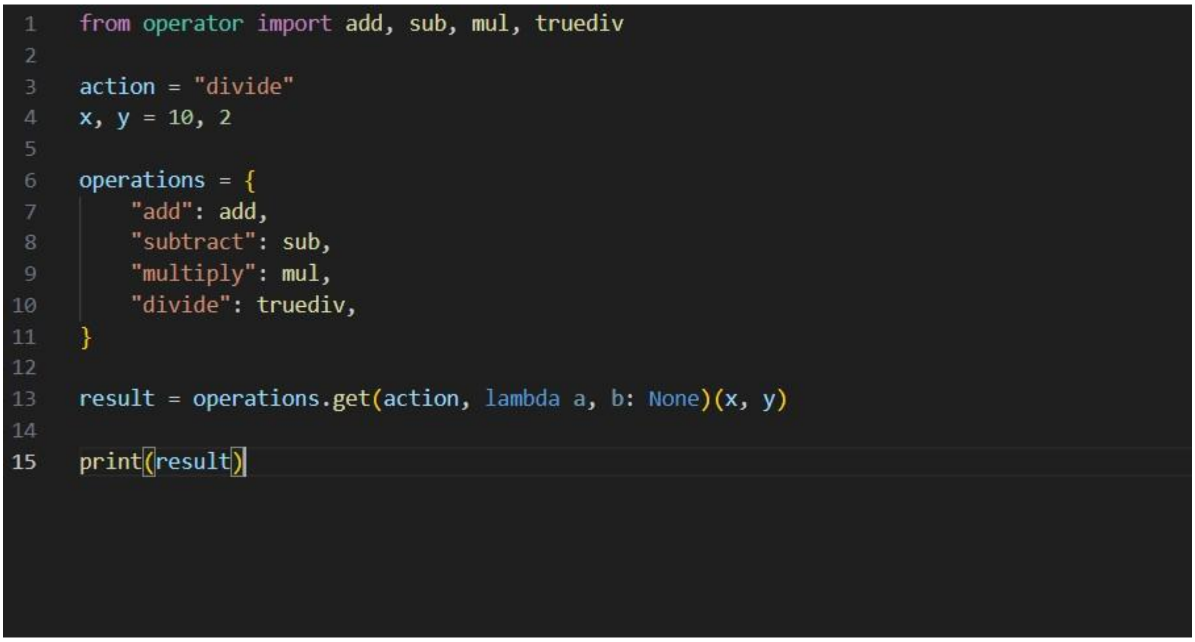
5.0

AI Prompt Used:

The following Python code uses multiple if-elif conditions to perform operations.
Refactor it into a cleaner and more scalable approach using mapping techniques or functions, without changing the output.

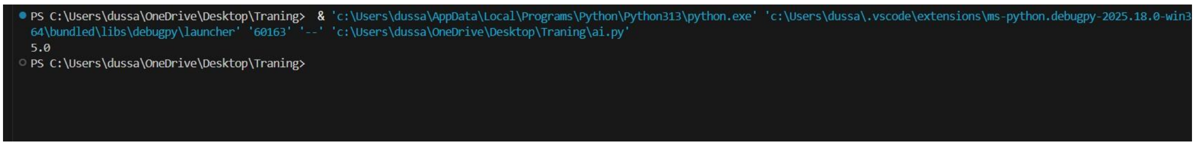
```
action = "divide"
x, y = 10, 2

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)
```

Code:



```python
from operator import add, sub, mul, truediv

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

operations = {
    "add": add,
    "subtract": sub,
    "multiply": mul,
    "divide": truediv,
}

result = operations.get(action, lambda a, b: None)(x, y)

print(result)
```

Output:



```
PS C:\Users\dussa\OneDrive\Desktop\Traning>  & 'c:\Users\dussa\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\dussa\.vscode\extensions\ms-python.debugpy-2025.18.0-win3
64\bundled\libs\debugpy\launcher' '60163' '--' 'c:\Users\dussa\OneDrive\Desktop\Traning\ai.py'
5.0
PS C:\Users\dussa\OneDrive\Desktop\Traning>
```

Task 5:

Simplifying Search Logic in Collections

Scenario

You are reviewing legacy inventory-check code that uses manual loops to find elements.

Task Description

- Identify the explicit loop used for searching an item.
- Use AI assistance to refactor the logic into a more concise and readable form.
- Maintain the same output behavior. **Legacy 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")

Expected Output

Item Available

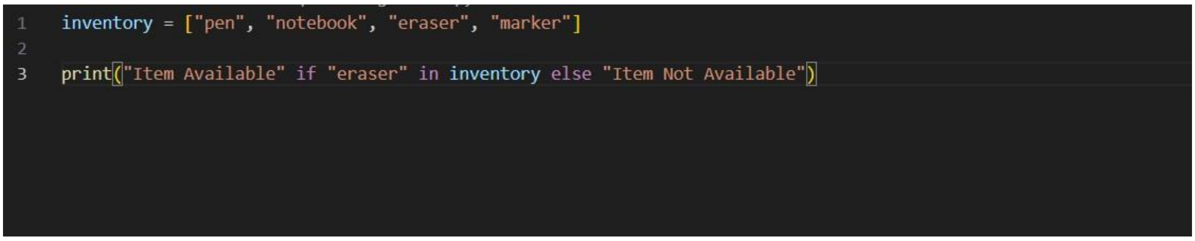
AI Prompt Used:

Refactor the following Python code to simplify the search logic.
Replace manual looping and flag variables with a more concise and Pythonic solution while keeping the same output behavior.

inventory = ["pen", "notebook", "eraser", "marker"] found
= False

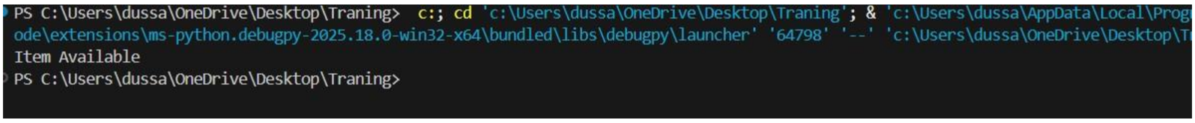
```
for item in inventory:
if item == "eraser":
        found = True
        break

print("Item Available" if found else "Item Not Available")
```

Code:



```
inventory = ["pen", "notebook", "eraser", "marker"]

print("Item Available" if "eraser" in inventory else "Item Not Available")
```

Output:



```
PS C:\Users\dussa\OneDrive\Desktop\Traning>  c:; cd 'c:\Users\dussa\OneDrive\Desktop\Traning'; & 'c:\Users\dussa\AppData\Local\Prog
ode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64798' '--' 'c:\Users\dussa\OneDrive\Desktop\T
Item Available
PS C:\Users\dussa\OneDrive\Desktop\Traning>
```