

AI ASSISTED CODING

NAME: E.HAMSITHA

ENROLL NUMBER: 2403A52361

BATCH NUMBER :13

Lab assignment-13.4

Task 1

- **Task:** Refactor repeated loops into a cleaner, more Pythonic approach.

Instructions:

- Analyze the legacy code.
- Identify the part that uses loops to compute values.
- Refactor using **list comprehensions** or helper functions while keeping the output the same.

Legacy Code:

```
numbers = [1, 2, 3, 4, 5]
```

```
squares = []
```

```
for n in numbers:
```

```
    squares.append(n ** 2)
```

```
print(squares)
```

```
[1]
✓ 0s # Legacy code (for reference)
# numbers = [1, 2, 3, 4, 5]
# squares = []
# for n in numbers:
#     squares.append(n ** 2)
# print(squares)

# ✓ Refactored code using list comprehension
numbers = [1, 2, 3, 4, 5]
squares = [n ** 2 for n in numbers]
print(squares)

... [1, 4, 9, 16, 25]
```

Task 2

Task: Simplify string concatenation.

Instructions:

- Review the loop that builds a sentence using +=.
- Refactor using " ".join() to improve efficiency and readability.

Legacy Code:

```
words = ["AI", "helps", "in", "refactoring", "code"]
```

```
sentence = ""
```

```
for word in words:
```

```
    sentence += word + " "
```

```
print(sentence.strip())
```

```
[2]
✓ Os
▶ # Legacy code (for reference)
# words = ["AI", "helps", "in", "refactoring", "code"]
# sentence = ""
# for word in words:
#     sentence += word + " "
# print(sentence.strip())

# ✅ Refactored code using " ".join()
words = ["AI", "helps", "in", "refactoring", "code"]
sentence = " ".join(words)
print(sentence)
```

⌵ ... AI helps in refactoring code

Task 3

Task: Replace manual dictionary lookup with a safer method.

Instructions:

- Check how the code accesses dictionary keys.
- Use `.get()` or another Pythonic approach to handle missing keys gracefully.

Legacy Code:

```
student_scores = {"Alice": 85, "Bob": 90}
```

```
if "Charlie" in student_scores:
```

```
    print(student_scores["Charlie"])
```

```
else:
```

```
    print("Not Found")
```

```
[3]
✓ Os
# Legacy code (for reference)
# student_scores = {"Alice": 85, "Bob": 90}
# if "Charlie" in student_scores:
#     print(student_scores["Charlie"])
# else:
#     print("Not Found")

# ✓ Refactored code using .get() for safer lookup
student_scores = {"Alice": 85, "Bob": 90}
print(student_scores.get("Charlie", "Not Found"))

... Not Found
```

Task 4

Task: Refactor repetitive if-else blocks.

Instructions:

- Examine multiple if-elif statements for operations.
- Refactor using **dictionary mapping** to make the code scalable and clean.

Legacy Code:

```
operation = "multiply"
```

```
a, b = 5, 3
```

```
if operation == "add":
```

```
    result = a + b
```

```
elif operation == "subtract":
```

```
    result = a - b
```

```
elif operation == "multiply":
```

```
    result = a * b
```

```
else:
```

```
result = None
```

```
print(result)
```



The screenshot shows a Jupyter Notebook cell with two code blocks. The first block is commented out and labeled '# Legacy code (for reference)'. It contains a function that checks the 'operation' variable and sets 'result' accordingly. The second block is labeled '# Refactored code using dictionary mapping' and shows a more concise implementation using a dictionary of lambda functions and the 'get' method. The cell is executed, as indicated by the green checkmark and '0s' in the top left corner.

```
[4] ✓ 0s # Legacy code (for reference)
# operation = "multiply"
# a, b = 5, 3
#
# if operation == "add":
#     result = a + b
# elif operation == "subtract":
#     result = a - b
# elif operation == "multiply":
#     result = a * b
# else:
#     result = None
#
# print(result)

# ✓ Refactored code using dictionary mapping
operation = "multiply"
a, b = 5, 3

operations = {
    "add": lambda x, y: x + y,
    "subtract": lambda x, y: x - y,
    "multiply": lambda x, y: x * y
}

result = operations.get(operation, lambda x, y: None)(a, b)
print(result)
```

Task 5

Task: Optimize nested loops for searching.

Instructions:

- Identify the nested loop used to find an element.
- Refactor using Python's in keyword or other efficient search techniques.

Legacy Code:

```
items = [10, 20, 30, 40, 50]
```

```
found = False
```

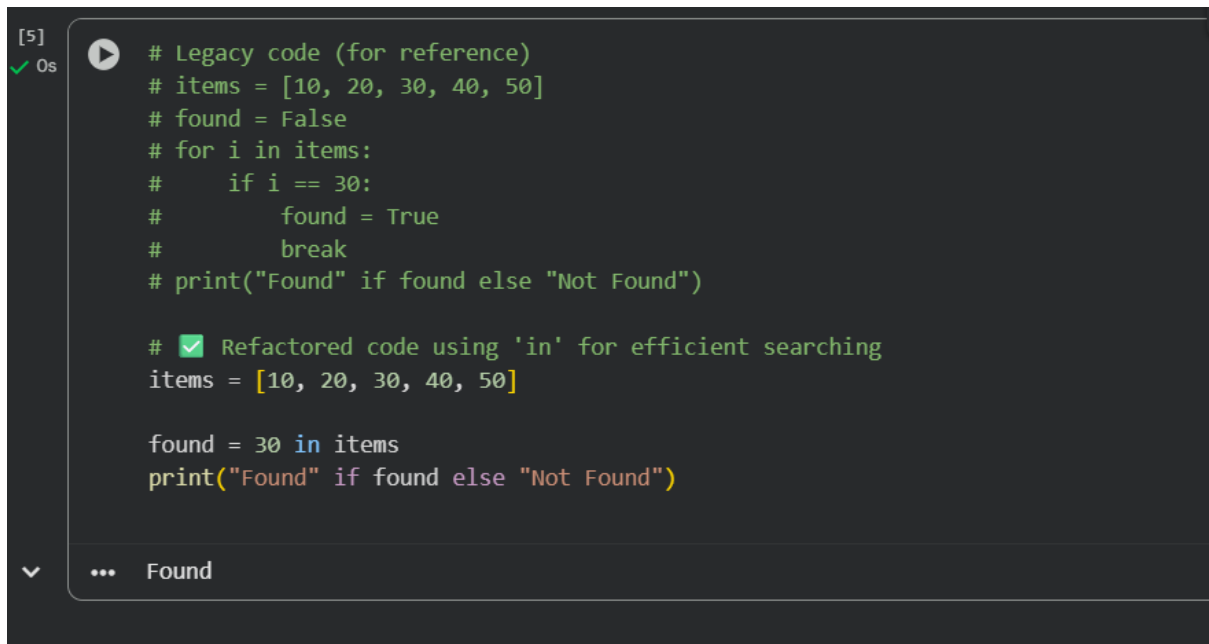
```
for i in items:
```

```
if i == 30:
```

```
    found = True
```

```
    break
```

```
print("Found" if found else "Not Found")
```



```
[5]  
✓ 0s  
# Legacy code (for reference)  
# items = [10, 20, 30, 40, 50]  
# found = False  
# for i in items:  
#     if i == 30:  
#         found = True  
#         break  
# print("Found" if found else "Not Found")  
  
# ✓ Refactored code using 'in' for efficient searching  
items = [10, 20, 30, 40, 50]  
  
found = 30 in items  
print("Found" if found else "Not Found")
```

Found