

LAB ASSIGNMENT – 13

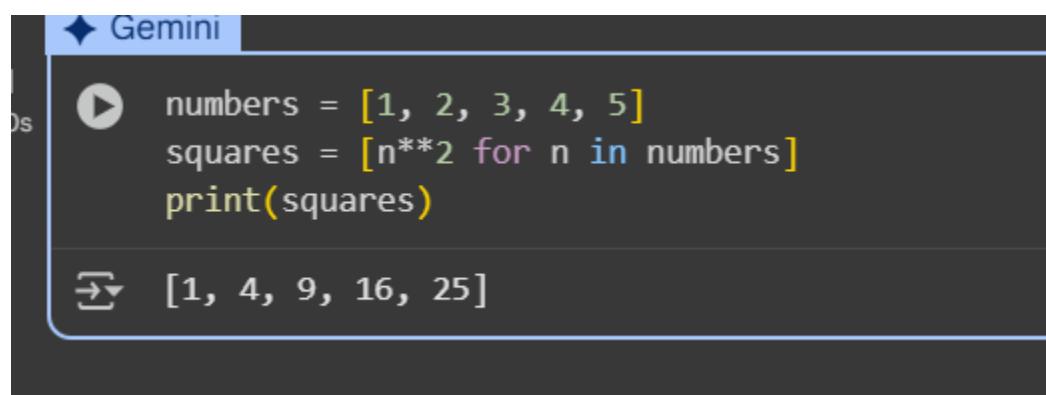
NAME : M.SHIVA
HALL.NO : 2403A52377
BATCH.NO : AI 14

Prompt 01:

Make this code shorter using list comprehension. Keep the same output.

```
numbers = [1, 2, 3, 4, 5]
squares = []
for n in numbers: squares.append(n ** 2)
print(squares)
```

code&Output:



The screenshot shows a code editor window titled "Gemini". The code area contains the following Python code:

```
numbers = [1, 2, 3, 4, 5]
squares = [n**2 for n in numbers]
print(squares)
```

Below the code, the output is displayed in a separate pane:

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

Code explanation:

This code first creates a list of numbers from 1 to 5 called numbers. Then, it uses a list comprehension [n^{**2} for n in numbers] to create a new list called squares. This list comprehension iterates through each number n in the numbers list and calculates its square (n^{**2}), adding the result to the squares list. Finally, it prints the squares list to the console.

Prompt 02:

Use ".join() to join the words instead of using +=. Output should be same.

```
words = ["AI", "helps", "in", "refactoring", "code"]  
sentence = ""  
for word in words:  
    sentence += word + " "  
print(sentence.strip())
```

Code&Output:

```
▶ words = ["AI", "helps", "in", "refactoring", "code"]  
      sentence = " ".join(words)  
      print(sentence)  
→ AI helps in refactoring code
```

Code explanation:

This code first creates a list of strings called words. Then, it uses the ".join(words) method to create a single string called sentence. The join() method concatenates all the elements in the words list into a single string, using the string it's called on (in this case, a space " ") as a separator between each element. Finally, it prints the resulting sentence to the console.

Prompt 03:

Change this code to use .get() for safe dictionary access. Output should not change.

```
student_scores = {"Alice": 85, "Bob": 90}  
if "Charlie" in student_scores:  
    print(student_scores["Charlie"])  
else:  
    print("Not Found")
```

Code:

```
▶ student_scores = {"Alice": 85, "Bob": 90}  
    print(student_scores.get("Charlie", "Not Found"))  
→ Not Found
```

Code explanation:

Okay, I will add the code to access the dictionary safely using .get() to the selected empty cell.

I've updated the cell to use .get() for safe dictionary access as you requested.

Would you like to explore other dictionary methods or perhaps work with a different data structure?

Prompt 04:

Replace if-else with a dictionary for operations. Keep the same output.

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)
```

Code:

```
▶ operation = "multiply"
  a, b = 5, 3

operations = {
    "add": a + b,
    "subtract": a - b,
    "multiply": a * b,
}

result = operations.get(operation)
print(result)
```

◀ 15

Code explanation:

This code uses a dictionary operations to store different arithmetic operations with keys like "add", "subtract", and "multiply". It then uses the .get() method with the operation variable as the key to retrieve the result of the desired operation. This is a clean way to select and perform actions based on a given input string, avoiding lengthy if-elif-else chains.

Prompt 05:

Use in keyword instead of loop to find the item. Output should be same.

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")
```

Code:

```
▶ items = [10, 20, 30, 40, 50]
  print("Found" if 30 in items else "Not Found")
→ Found
```

Code explanation:

This code checks if the number 30 is present in the items list using the `in` keyword. If it is found, it prints "Found"; otherwise, it prints "Not Found". This is a more Pythonic and efficient way to check for membership compared to iterating through the list with a loop.