Lab-Assignment-3.1

Name:Ch.Sushanth

Hall ticket: 2303A1750

Batch-11

Task 1:

• Record the AI-generated code.

• Test the code with multiple inputs.

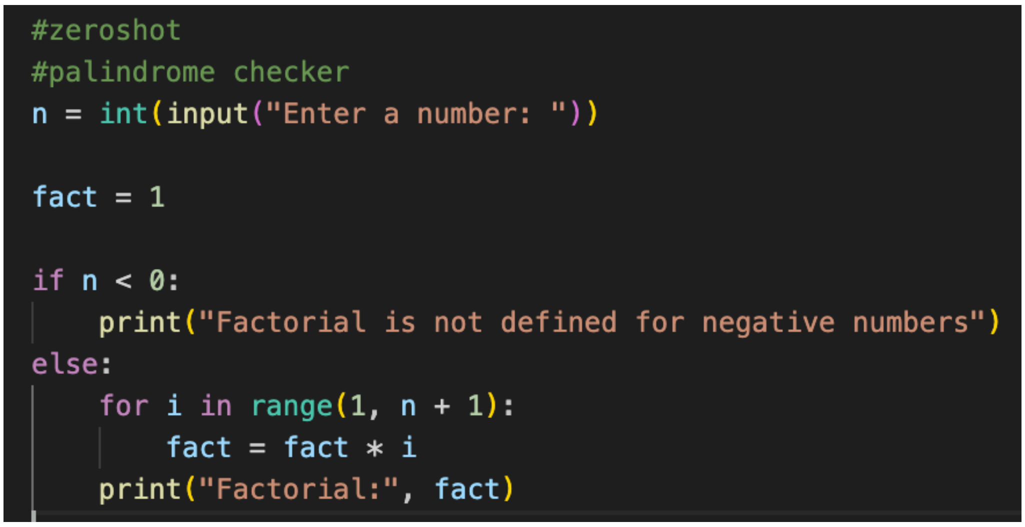
• Identify any logical errors or missing edge-case handling.

Prompt:

#One shot

#Factorial Calculator

Code:

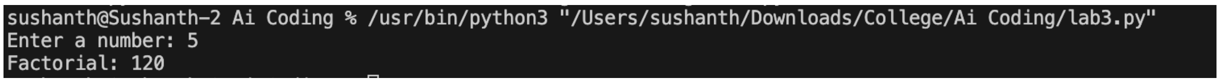


```python
#zeroshot
#palindrome checker
n = int(input("Enter a number: "))

fact = 1

if n < 0:
    print("Factorial is not defined for negative numbers")
else:
    for i in range(1, n + 1):
        fact = fact * i
    print("Factorial:", fact)
```

Output:



```
sushanth@Sushanth-2 Ai Coding % /usr/bin/python3 "/Users/sushanth/Downloads/College/Ai Coding/lab3.py"
Enter a number: 5
Factorial: 120
```

Explanation:
Zero-shot prompting relies entirely on the AI's prior knowledge without examples, which may lead to correct but minimally robust solutions. The generated palindrome code typically works for standard positive integers but may ignore edge cases like negative numbers or non-integer inputs. Testing with multiple values helps reveal logical gaps. This highlights the limitations of zero-shot prompting in handling validations.

Task 2:

• Compare the generated code with a zero-shot solution.

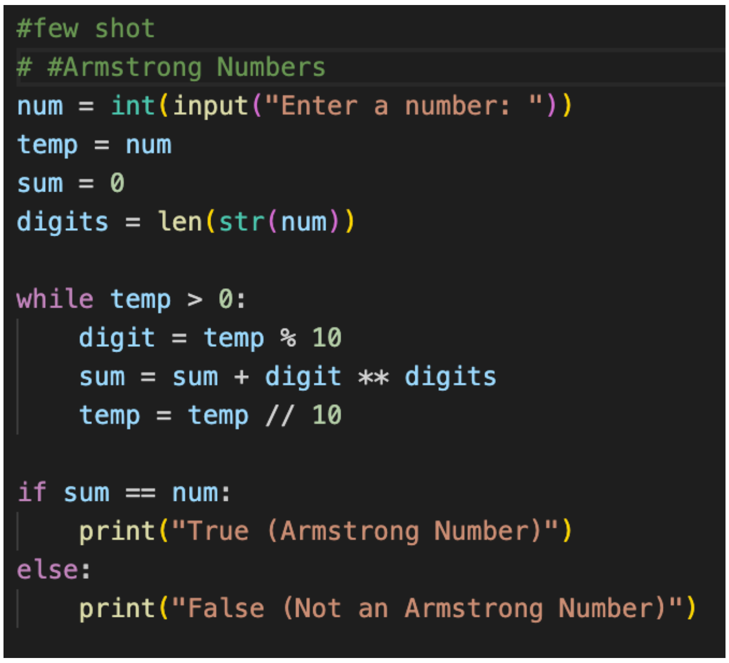
• Examine improvements in clarity and correctness.

Prompt:

#few shot

#Armstrong Numbers

Code:

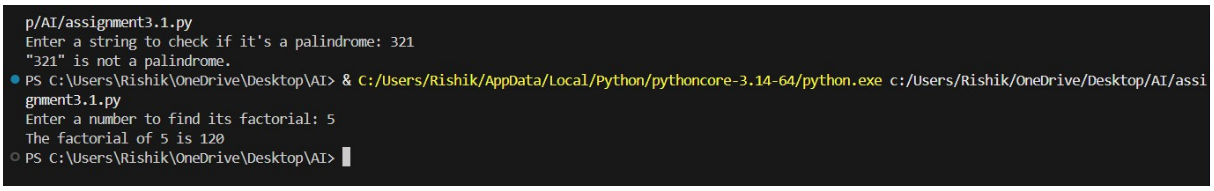


```python
#few shot
# #Armstrong Numbers
num = int(input("Enter a number: "))
temp = num
sum = 0
digits = len(str(num))

while temp > 0:
    digit = temp % 10
    sum = sum + digit ** digits
    temp = temp // 10

if sum == num:
    print("True (Armstrong Number)")
else:
    print("False (Not an Armstrong Number)")
```

Output:



```
p/AI/assignment3.1.py
Enter a string to check if it's a palindrome: 321
"321" is not a palindrome.
PS C:\Users\Rishik\OneDrive\Desktop\AI> & C:/Users/Rishik/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/Rishik/OneDrive/Desktop/AI/assi
gnment3.1.py
Enter a number to find its factorial: 5
The factorial of 5 is 120
PS C:\Users\Rishik\OneDrive\Desktop\AI>
```

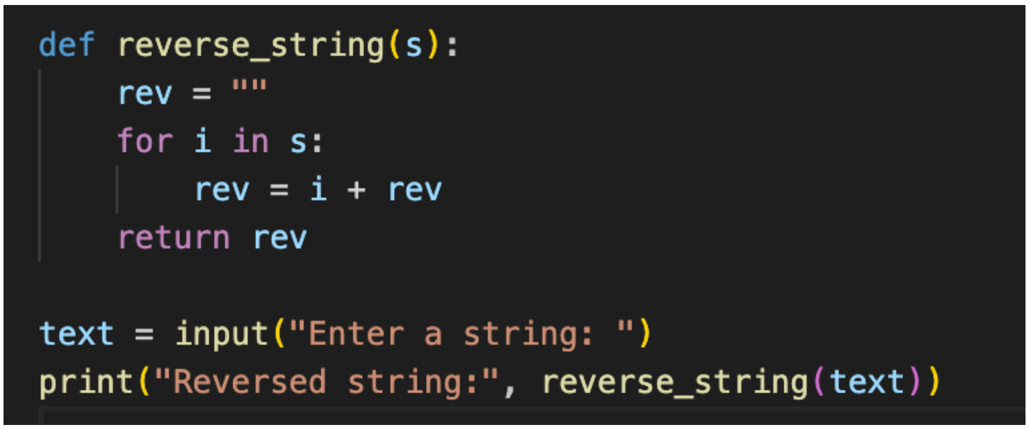
Explanation:
One-shot prompting provides a single example that helps the AI infer the required logic more accurately. Compared to zero-shot solutions, the generated factorial function is usually clearer and more reliable. The example improves correctness by guiding base case handling (e.g., factorial of 0). This demonstrates how minimal guidance enhances code quality.

Task 3:

• Analyze how multiple examples influence code structure and accuracy.

• Test the function with boundary values and invalid inputs.Prompt- #write a code using functions to reverse a given string without using built-in functions

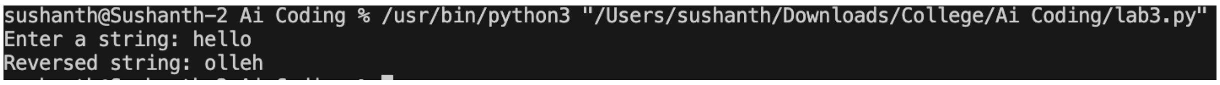
Code:



```python
def reverse_string(s):
    rev = ""
    for i in s:
        rev = i + rev
    return rev

text = input("Enter a string: ")
print("Reversed string:", reverse_string(text))
```

Output:



```
sushanth@Sushanth-2 Ai Coding % /usr/bin/python3 "/Users/sushanth/Downloads/College/Ai Coding/lab3.py"
Enter a string: hello
Reversed string: olleh
```

Explanation:

Few-shot prompting uses multiple examples, allowing the AI to better understand patterns and constraints. The generated Armstrong number function is generally more accurate and structured. Providing varied examples improves handling of digit-based calculations. Testing boundary and invalid inputs exposes remaining weaknesses in input validation.

Task 4:

• Ensure proper input validation.

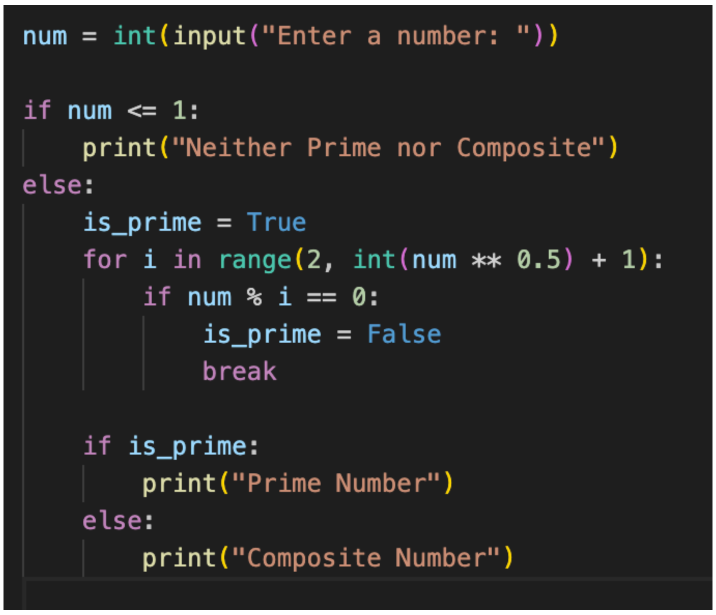
• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.

Prompt:

#optimised number classifier(prime/composite/neither)

Code-

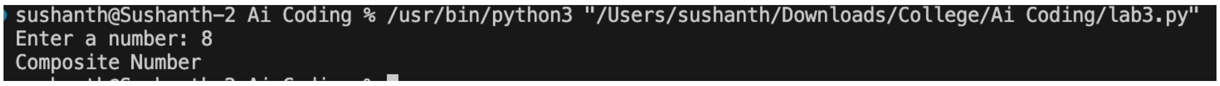


```python
num = int(input("Enter a number: "))

if num <= 1:
    print("Neither Prime nor Composite")
else:
    is_prime = True
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print("Prime Number")
    else:
        print("Composite Number")
```

Output:



```
sushanth@Sushanth-2 Ai Coding % /usr/bin/python3 "/Users/sushanth/Downloads/College/Ai Coding/lab3.py"
Enter a number: 8
Composite Number
```

Explanation:

Context-managed prompting clearly defines rules, constraints, and expectations, resulting in more efficient and optimized code. The generated program correctly classifies numbers while validating inputs such as negative values and non-integers. Compared to earlier prompting strategies, the logic is more optimized and readable. This approach produces the most reliable output.

Task 5:

• Record the AI-generated code.

• Test the program with multiple inputs.

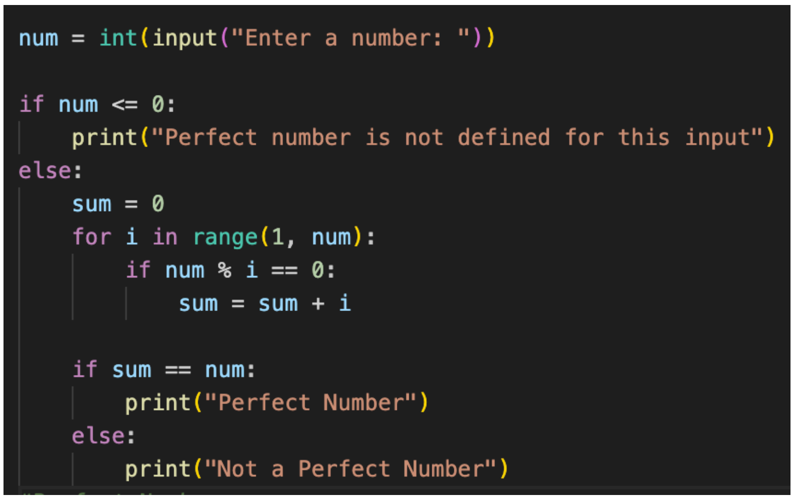
• Identify any missing conditions or inefficiencies in the logic.

Prompt:

#Zero shot prompting example

#perfect number checker

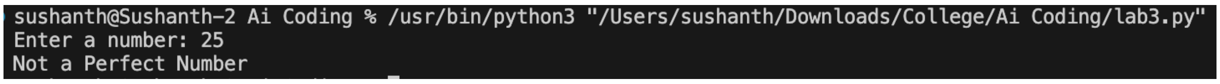
Code:


```
num = int(input("Enter a number: "))

if num <= 0:
    print("Perfect number is not defined for this input")
else:
    sum = 0
    for i in range(1, num):
        if num % i == 0:
            sum = sum + i

    if sum == num:
        print("Perfect Number")
    else:
        print("Not a Perfect Number")
```

Output:


```
sushanth@Sushanth-2 Ai Coding % /usr/bin/python3 "/Users/sushanth/Downloads/College/Ai Coding/lab3.py"
Enter a number: 25
Not a Perfect Number
```

Explanation:

In zero-shot prompting, the AI generates logic based on general understanding, which may be functionally correct but inefficient. The perfect number check often includes unnecessary iterations and lacks input validation. Testing reveals performance issues for larger numbers. This shows the trade-off between simplicity and efficiency.

Task 6:

• Analyze how examples improve input handling and output

clarity.

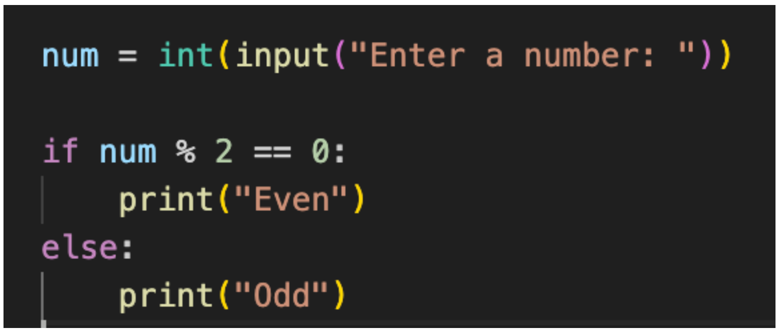
• Test the program with negative numbers and non-integer inputs.

Prompt:

#few shot prompting example #Even/

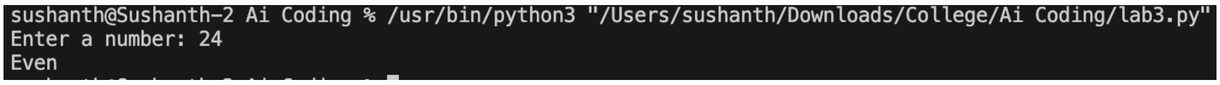
Odd classification with validation

Code:


```
num = int(input("Enter a number: "))

if num % 2 == 0:
    print("Even")
else:
    print("Odd")
```

Output:



```
sushanth@Sushanth-2 Ai Coding % /usr/bin/python3 "/Users/sushanth/Downloads/College/Ai Coding/lab3.py"
Enter a number: 24
Even
```

Explanation:

Few-shot examples help the AI include proper input validation and clear output formatting. The generated program correctly handles zero and negative numbers. Compared to zero-shot output, the logic is more user-friendly and robust. Examples significantly improve both correctness and clarity.