

AI Assisted Coding

Assignment-03

Name : Abhiram Yadav

Hallticket:2303A51086

Batch-02

Question:01

- A working Python program generated with Copilot assistance

```
Palindrome.py > ...
1 #Assignment_3
2 #2303A51086
3 #Date: 19-01-2026
4 #write a python program to check whether a given number is a palindrome or not
5 number = int(input("Enter a number to check if it is a palindrome: "))
6 original_number = number
7 reversed_number = 0
8 while number > 0:
9     digit = number % 10
10    reversed_number = reversed_number * 10 + digit
11    number //= 10
12 if original_number == reversed_number:
13     print(f"{original_number} is a palindrome.")
14 else:
15     print(f"{original_number} is not a palindrome.")
16
```

- Sample input/output screenshots

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Enter a number to check if it is a palindrome: 121
121 is a palindrome.

PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> 12332
12332

PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/OneDrive/Local/Programs/Python/Python311/p
/AI ASSISTED 3-2/Palindrome.py"
Enter a number to check if it is a palindrome: 12332
12332 is not a palindrome.
```

Logical Errors & Missing Edge-Case Handling

Issue 1: Input 0

- **Problem:**

The while number > 0 loop never runs for 0.

- **Result:**

reversed_number remains 0, so output is correct **by coincidence**, not logic.

- **Fix Needed:**
Explicitly handle 0.
-

Issue 2: Negative Numbers

- **Problem:**
Negative numbers are not handled.
 - **Example:**
 $-121 \rightarrow$ loop doesn't run \rightarrow reversed number stays 0.
 - **Expected Behavior:**
Negative numbers should **not** be palindromes.
-

Issue 3: Not a Function

- **Problem:**
The prompt asked for a **function**, but Copilot generated a **script**.
- **Impact:**
Less reusable and not ideal for testing or modular programming.

Edge-Case Handling generated code :

```
def is_palindrome(number):
    if number < 0:
        return False

    original = number
    reversed_num = 0

    while number > 0:
        digit = number % 10
        reversed_num = reversed_num * 10 + digit
        number //= 10

    return original == reversed_num

# Testing
print(is_palindrome(121))  # True
print(is_palindrome(123))  # False
print(is_palindrome(0))    # True
print(is_palindrome(-121)) # False
```

Question-02

- A working Python program generated with Copilot assistance

```
#Assignment 3
#2303A51086
#Date: 19-01-2026
...
a=5
display the factorial of a is 120
...
number = int(input("Enter a number to check factorial: "))
factorial = 1
for i in range(1, number + 1):
    factorial *= i
print(f"The factorial of {number} is {factorial}")
```

- Screenshot(s) showing: Input/Output

```
/AI ASSISTED 3-2/Sample.py"
Enter a number to check factorial: 5
The factorial of 5 is 120
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/DELL/AppData/Local/Programs/Python/Python311/py
/AI ASSISTED 3-2/Sample.py"
Enter a number to check factorial: 6
The factorial of 6 is 720
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> |
```

side by side comparition

Zero-Shot:

```
#Assignment 3
#2303A51086
#Date: 19-01-2026
# Write a python code to print the factorial of a given number
number = int(input("Enter a number to find its factorial: "))
factorial = 1
for i in range(1, number + 1):
    factorial *= i
print(f"The factorial of {number} is {factorial}")
```

One-Shot:

```
#Assignment 3
#2303A51086
#Date: 19-01-2026
...
a=5
display the factorial of a is 120
...
number = int(input("Enter a number to check factorial: "))
factorial = 1
for i in range(1, number + 1):
    factorial *= i
print(f"The factorial of {number} is {factorial}")
```

Aspect	Zero-Shot Prompt	One-Shot Prompt
Example provided	No	Yes ($5 \rightarrow 120$)
Code structure	Basic loop	Same loop
Clarity of intent	Moderate	Better
Correct output	Yes	Yes
Logic correctness	Correct	Correct
Readability	Normal	Improved with explanation

Improvements Observed in One-Shot Prompting

Clearer understanding of requirement

The example ($5 \rightarrow 120$) helps the AI clearly understand the expected behavior.

Better documentation

The AI included a comment block explaining the example.

No logical change needed

The factorial logic was already correct, so one-shot prompting mainly improved **clarity**, not logic.

Question-03

Few-Shot Prompting (Armstrong Number Check)

Code given by copilot :

```
#Assignment 3
#2303A51086
#Date: 19-01-2026
...
a=153
display it as a Armstrong Number
a=370
display it as a Armstrong Number
a=123
display it as Not an Armstrong Number

...
number = int(input("Enter a number to check if it's an Armstrong number: "))
num_str = str(number)
num_digits = len(num_str)
armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
if armstrong_sum == number:
    print(f"{number} is an Armstrong Number")
else:
    print(f"{number} is Not an Armstrong Number")
```

➤ Sample input/output screenshots

```
/AI ASSISTED 3-2/Sample.py"
Enter a number to check if it's an Armstrong number: 153
153 is an Armstrong Number
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/OneDrive/AppData/Local/Programs/Python/Python311/python.exe "
/AI ASSISTED 3-2/Sample.py"
Enter a number to check if it's an Armstrong number: 123
123 is Not an Armstrong Number
PS C:\Users\OneDrive\Documents\RAM\AI ASSISTED 3-2>
```

Ln 20, Col 50 S

Influence of Few-Shot Prompting on Code Structure & Accuracy

Correct logic selection

Because multiple examples were given, the AI correctly:

- Converted the number to a string
- Counted digits dynamically
- Used power based on number of digits

Generalized solution

The code works not only for 3-digit numbers but also for:

- 1-digit Armstrong numbers
- Higher-digit Armstrong numbers

Improved clarity

The examples helped the AI understand the expected output format clearly.

Testing with Boundary Values

Input	Expected Output	Actual Output	Correct?
0	Armstrong Number	Armstrong Number	✓
1	Armstrong Number	Armstrong Number	✓
9	Armstrong Number	Armstrong Number	✓
153	Armstrong Number	Armstrong Number	✓
9474	Armstrong Number	Armstrong Number	✓
10	Not Armstrong	Not Armstrong	✓

Testing with Invalid Inputs

Input	Result	Issue
-153	Error / Incorrect	✗ Negative numbers not handled
abc	Program crash	✗ No input validation
12.5	Program crash	✗ Integer-only handling

Question 4:

Context-Managed Prompting (Optimized Number Classification)

Code given by copilot :

```
Sample.py > ...
1  #Assignment 3
2  #2303A51086
3  #Date: 19-01-2026
4  ...
5  Task: Write an optimized Python program that classifies a given number as Prime, Composite, or Neither.
6
7  Instructions & Constraints:
8
9  1.Accept input from the user and validate that it is an integer.
10
11 2.If the number is less than or equal to 1, classify it as Neither.
12
13 3.Use an efficient algorithm with time complexity O( $\sqrt{n}$ ) to check primality.
14
15 4.Avoid unnecessary iterations and redundant checks.
16
17 5.Follow clean coding practices and include brief comments.
18
19  Output Requirements:
20
21 1.Print "Prime" if the number is prime.
22
23 2.Print "Composite" if the number is composite.
24
25 3.Print "Neither" if the number is 0, 1, or negative.
26
27 Note: Ensure the program handles invalid inputs gracefully.
```

```
Sample.py > ...
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

➤ Sample input/output screenshots

```
/AI ASSISTED 3-2/Sample.py"
Enter an integer: 23
Prime
PS C:\Users\Dell\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/Python/Python311/python.exe /AI ASSISTED 3-2/Sample.py"
Enter an integer: 0
Neither
PS C:\Users\Dell\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/Dell/AppData/Local/Programs/Python/Python311/python.exe /AI ASSISTED 3-2/Sample.py"
Ln 52, Col 1
```

How the Program Meets the Task Requirements

Proper Input Validation

- Uses try-except to handle non-integer inputs.
- Prevents program crashes due to invalid user input.

Optimized Logic for Efficiency

- Checks divisibility only up to \sqrt{n} , reducing time complexity from $O(n)$ to $O(\sqrt{n})$.
- Stops early when a factor is found.

Correct Classification

- $\text{num} \leq 1 \rightarrow \text{Neither}$
- Divisible by any number $\rightarrow \text{Composite}$
- Otherwise $\rightarrow \text{Prime}$

Comparison with Earlier Prompting Strategies

<u>Aspect</u>	<u>Zero-Shot Prompting</u>	<u>Few-Shot Prompting</u>	<u>Context-Managed Prompting</u>
<u>Instruction Clarity</u>	<u>Low</u>	<u>Medium</u>	<u>High</u>
<u>Input Validation</u>	<u>Often missing</u>	<u>Sometimes included</u>	<u>Explicitly enforced</u>
<u>Algorithm Efficiency</u>	<u>$O(n)$ loops</u>	<u>Partial optimization</u>	<u>Guaranteed $O(\sqrt{n})$</u>
<u>Edge Case Handling</u>	<u>Weak</u>	<u>Partial</u>	<u>Complete</u>
<u>Code Quality</u>	<u>Inconsistent</u>	<u>Improved</u>	<u>Clean & professional</u>

Question 5:

Zero-Shot Prompting (Perfect Number Check)

Code given by copilot :

```
Sample.py > ...
1  #Assignment 3
2  #2303A51086
3  #Date: 19-01-2026
4  ...
5  Task: Write an optimized Python program that classifies a given number as Prime, Composite, or Neither.
6
7  Instructions & Constraints:
8
9  1.Accept input from the user and validate that it is an integer.
10
11 2.If the number is less than or equal to 1, classify it as Neither.
12
13 3.Use an efficient algorithm with time complexity O( $\sqrt{n}$ ) to check primality.
14
15 4.Avoid unnecessary iterations and redundant checks.
16
17 5.Follow clean coding practices and include brief comments.
18
19  Output Requirements:
20
21 1.Print "Prime" if the number is prime.
22
23 2.Print "Composite" if the number is composite.
24
25 3.Print "Neither" if the number is 0, 1, or negative.
26
27 Note: Ensure the program handles invalid inputs gracefully.
```

```
Sample.py > ...
1  #Assignment 3
2  #2303A51086
3  #Date: 19-01-2026
4  ...
5  write a python program to whether a number is perfect number or not.
6
7  ...
8  num=int(input("Enter a number: "))
9  sum=0
10 for i in range(1,num):
11     if num%i==0:
12         sum=sum+i
13 if sum==num:
14     print(f"{num} is a perfect number")
15 else:
16     print(f"{num} is not a perfect number")
17 #Time Complexity: O(n)
18 #Space Complexity: O(1)
```

➤ Sample input/output screenshots

```
/AI ASSISTED 3-2/Sample.py"
Enter a number: 121
121 is not a perfect number
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/DELL/AppData/Local/Programs/Py
/AI ASSISTED 3-2/Sample.py"
Enter a number: 6
6 is a perfect number
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2>
```

Testing the Program with Multiple Inputs

Input Expected Output Actual Output

6	Perfect Number	True
28	Perfect Number	True
12	Not Perfect	False
1	Not Perfect	False
0	Not Perfect	False

Missing Conditions and Inefficiencies

Missing Conditions

- No check for **negative numbers or zero** (perfect numbers are defined only for positive integers).
- No input validation for non-integer values.

Inefficiencies

- The loop runs from 1 to num-1, giving **O(n)** time complexity.
- Checking divisors beyond num/2 is unnecessary.
- Can be optimized to **O(√n)** by checking divisor pairs.

Comparison Summary

Aspect	Zero-Shot Output	Optimized Version
Input Validation	Missing	Added
Edge Case Handling	Weak	Proper
Time Complexity	$O(n)$	$O(\sqrt{n})$
Efficiency	Low	High
Code Quality	Basic	Improved

Question 6:

Few-Shot Prompting (Even or Odd Classification with Validation)

Code given by copilot :

```
Sample.py > ...
1 #Assignment 3
2 #2303A51086
3 #Date: 19-01-2026
4 ...
5 a=8
6 display a as an Even number
7 a=15
8 display a as an odd number
9 a=0
10 display a as an Even number
11 ...
12 a = int(input("Enter a number: "))
13 if a % 2 == 0:
14     print(f"{a} is an Even number")
15 else:
16     print(f"{a} is an Odd number")
17 #End of the program
```

➤ Sample input/output screenshots

```
/AI ASSISTED 3-2/Sample.py"
Enter a number: 6
6 is a perfect number
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> & C:/Users/DELL/AppData/Local/Programs/Python
/AI ASSISTED 3-2/Sample.py"
Enter a number: 45
45 is an Odd number
PS C:\Users\DELL\OneDrive\Documents\RAM\AI ASSISTED 3-2> []
```

Testing with Different Inputs

Input Output

8	Even
15	Odd
0	Even
-4	Even

Input Output

-7 Odd

abc Invalid input (in improved version)

3.5 Invalid input (in improved version)

Analysis: How Few-Shot Examples Improve the Program

Improved Input Handling

- Examples clearly show that 0 should be treated as **Even**.
- Negative numbers follow the same modulus logic.
- Encourages inclusion of input validation using try-except.

Improved Output Clarity

- Output format becomes consistent (Even / Odd).
- Reduces ambiguity compared to zero-shot prompting.
- Helps the AI match expected output exactly.

Comparison with Zero-Shot Prompting

Aspect	Zero-Shot Prompting	Few-Shot Prompting
Guidance	Minimal	Clear via examples
Output Accuracy	Inconsistent	Consistent
Edge Case Handling	Often missed	Explicit (0 case)
Input Validation	Rare	Encouraged
Code Quality	Basic	Improved