

ASSIGNMENT – 3.1

NAME : S.RAJESH

HT.NO : 2303A52301

BATCH : 35

Question 1: Zero-Shot Prompting (Palindrome Number Program)

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

Task:

- Record the AI-generated code.
- Test the code with multiple inputs.
- Identify any logical errors or missing edge-case handling.

PROMPT : check whether the given number is palindrome or not in python.

The screenshot shows a code editor interface with a Python file named `Assignment_3.1.py` open. The code checks if a number is a palindrome without using functions. An AI assistant is prompting the user to check if a number is a palindrome, and the AI has generated a solution. The solution handles negative numbers, uses slice notation for string reversal, and returns `True` or `False`. Key features listed include handling negative numbers, including test cases, and having an interactive input section.

```
# Simple palindrome check without functions
# Get number from user
num = int(input("Enter a number: "))

# Convert to string to check
num_str = str(abs(num))

# Check if palindrome by comparing with reverse
if num_str == num_str[::-1]:
    print(f"{num} is a palindrome")
else:
    print(f"{num} is not a palindrome")
```

This screenshot is similar to the first one but includes a `TERMINAL` tab at the bottom. The terminal shows the execution of the `Assignment_3.1.py` script. It prompts the user for a number and prints whether it is a palindrome or not. The AI assistant continues to interact with the user about the palindrome check.

```
# Simple palindrome check without functions
# Get number from user
num = int(input("Enter a number: "))

# Convert to string to check
num_str = str(abs(num))

# Check if palindrome by comparing with reverse
if num_str == num_str[::-1]:
    print(f"{num} is a palindrome")
else:
    print(f"{num} is not a palindrome")
```

PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/UNI/SEM - VI/AI Assistant Coding/Assignment_3.1.py"
Enter a number: 12321
12321 is a palindrome
PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/UNI/SEM - VI/AI Assistant Coding/Assignment_3.1.py"
Enter a number: 66553
66553 is not a palindrome
PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding>

Question 2: One-Shot Prompting (Factorial Calculation)

Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

Input: 5 → Output: 120

Task:

- Compare the generated code with a zero-shot solution.
- Examine improvements in clarity and correctness.

PROMPT : Generate a Python function to compute the factorial of a given number.

Example:

Input: 5 → Output: 120

The screenshot shows a software interface for AI-assisted coding. On the left, a code editor displays a Python script named 'Assignment_3.1.py'. The script contains code for a palindrome check and a factorial calculator. On the right, a 'CHAT' window titled 'PALINDROME CHECK FOR A NUMBER IN PYTHON' shows the following interaction:

```
generate a Python function to compute the factorial of a given number.  
Example:  
Input: 5 → Output: 120
```

The AI has responded with a task summary and a note about adding a factorial function. Below this, a list of bullet points specifies the requirements for the factorial function:

- Takes a number and multiplies all integers from 2 down to 1
- Handles edge cases (0, 1, negative numbers)
- Includes test cases showing: factorial(5) = 120

At the bottom of the interface, there is a status bar showing file statistics and system information like battery level and network status.

The screenshot shows a code editor with a Python file named `Assignment_3.1.py`. The code defines a factorial function that handles negative numbers, edge cases (0 and 1), and prints examples. The terminal below shows the script running and calculating the factorial of 12321.

```

Assignment_3.1.py
Assignment_3.1.py >...
n = int(input("Enter a number for factorial: "))
if n < 0:
    print("Factorial not defined for negative numbers")
elif n == 0 or n == 1:
    print(f"Factorial of {n} = 1")
else:
    result = 1
    for i in range(2, n + 1):
        result *= i
    print(f"Factorial of {n} = {result}")
# Example: Input 5 → Output 120
print("\nExample: Factorial of 5 = 120")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - ×
PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding> & C:\Users\rishi\A
ppData\Local\Programs\Python\Python312\python.exe "c:/Users/rishi/OneDrive/Desktop/UNI/
SEM - VI/AI Assistant Coding/Assignment_3.1.py"
● SEM - VI/AI Assistant Coding/Assignment_3.1.py*
Enter a number: 12321
Enter a number: 12321
--- Factorial Calculation ---
Enter a number for factorial: 10
Factorial of 10 = 3628800
Example: Factorial of 5 = 120
PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding>

```

The AI assistant interface on the right shows a prompt for generating a Python function to compute the factorial of a given number, with an example input of 5 and output of 120. It also shows a message from Claude Haiku 4.5, a note about reading the Python file, and a summary of the task requirements.

Question 3: Few-Shot Prompting (Armstrong Number Check)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

Examples:

- Input: 153 → Output: Armstrong Number
- Input: 370 → Output: Armstrong Number
- Input: 123 → Output: Not an Armstrong Number

Task:

- Analyze how multiple examples influence code structure and accuracy.
- Test the function with boundary values and invalid inputs.

PROMPT : Python function to check whether a given number is an Armstrong number.

Examples:

- Input: 153 → Output: Armstrong Number
- Input: 370 → Output: Armstrong Number
- Input: 123 → Output: Not an Armstrong Number

The screenshot shows a coding environment with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Title Bar:** Assignment_3.1.py
- Code Area:**

```

35 # TASK - III
36 # Armstrong Number check without functions
37 # Armstrong number: sum of digits raised to power of number of digits
38
39 print("\n--- Armstrong Number Check ---")
40 num = int(input("Enter a number: "))
41
42 # Convert to string to get digits
43 num_str = str(num)
44 num_digits = len(num_str)
45
46 # Calculate sum of each digit raised to power of total digits
47 sum_of_powers = 0
48 for digit in num_str:
49     sum_of_powers += int(digit) ** num_digits
50
51 # Check if Armstrong number
52 if sum_of_powers == num:
53     print(f"{num} is an Armstrong Number")
54 else:
55     print(f"{num} is Not an Armstrong Number")
56
57 # Examples
58 print("\nExamples:")
59 print("153 is an Armstrong Number (1^3 + 5^3 + 3^3 = 153)")
60 print("370 is an Armstrong Number (3^3 + 7^3 + 0^3 = 370) Keep Undo 1 of 1 ↑ ↓")
61 print("123 is Not an Armstrong Number (1^3 + 2^3 + 3^3 = 36) / 9474 is an Armstrong Number (9^4 + 4^4 + 7^4 + 4^4 = 9474)
    
```
- Right Panel:**
 - CHAT:** PALINDROME CHECK FOR A NUMBER IN PYTHON
 - Examples:**
 - Input: 153 → Output: Armstrong Number
 - Input: 370 → Output: Armstrong Number
 - Input: 123 → Output: Not an Armstrong Number
 - How It Works:**
 - Gets the number of digits
 - Raises each digit to that power and sums them
 - If the sum equals the original number, it's an Armstrong number
 - Examples: $153 = 1^3 + 5^3 + 3^3$, $370 = 3^3 + 7^3 + 0^3$
- Bottom Status Bar:** Line 63, Col 1, Spaces: 4, UTF-8, CRLF, Python 3.12 (64-bit), 08:09 PM, 12-01-2026

The screenshot shows a code editor with a Python file named `Assignment_3.1.py`. The code implements a function to check if a number is Armstrong. It also includes examples of Armstrong numbers and non-Armstrong numbers. To the right, an AI Assistant Coding interface displays a task titled "PALINDROME CHECK FOR A NUMBER IN PYTHON". It provides instructions, examples, and a summary of the completed task.

```
31
32     # Example: Input 5 → Output 120
33     print("\nExample: Factorial of 5 = 120")
34
35     # TASK - III
36     # Armstrong Number check without functions
37     # Armstrong number: sum of digits raised to power of number of digits
38
39     print("\n--- Armstrong Number Check ---")
40
41     num = int(input("Enter a number: "))
42
43     # Convert to string to get digits
44     num_str = str(num)
45     num_digits = len(num_str)

--- Armstrong Number Check ---
Enter a number: 234
234 is Not an Armstrong Number

Examples:
153 is an Armstrong Number (1^3 + 5^3 + 3^3 = 153)
370 is an Armstrong Number (3^3 + 7^3 + 0^3 = 370)
123 is Not an Armstrong Number (1^3 + 2^3 + 3^3 = 36)
```

(Optional Extension)

Question 4: Context-Managed Prompting (Optimized Number Classification)

Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

PROMPT : Write a Python program to classify a number as prime, composite, or neither. Ensure proper input validation and handle edge cases like 0, 1, and negatives. Use an optimized logic by checking factors only up to \sqrt{n} .

The screenshot shows the AI Assistant Coding interface. On the left, the code editor displays a Python file named Assignment_3.1.py. The code handles edge cases for negative numbers, 0, and 1. It then checks for primality by testing factors up to \sqrt{n} . If a factor is found, it prints "Composite". Otherwise, it prints "Prime". The right side of the interface shows a chat window with the task prompt and a summary of key features and optimizations.

```

65
66 print("\n--- Prime, Composite, or Neither ---")
67
68 num = int(input("Enter a number: "))
69
70 # Handle edge cases
71 if num < 0:
72     print(f"{num} is Neither (negative numbers are neither prime nor composite)")
73 elif num == 0 or num == 1:
74     print(f"{num} is Neither (0 and 1 are neither prime nor composite)")
75 elif num == 2:
76     print(f"{num} is Prime (only even prime number)")
77 else:
78     # Check if prime by testing factors up to sqrt(n)
79     is_prime = True
80     i = 2
81
82     while i * i <= num:
83         if num % i == 0:
84             is_prime = False
85             break
86         i += 1
87
88     if is_prime:
89         print(f"{num} is Prime")
90     else:
91         print(f"{num} is Composite")
92
93

```

Key features:

- Edge cases handled: negatives, 0, 1 → classified as "Neither"
- Special case: 2 is Prime (only even prime)
- Optimized logic: Only checks factors up to \sqrt{n} using `while i * i <= num`
- Input validation: Classifies any number correctly as Prime, Composite, or Neither

This approach is efficient because checking up to \sqrt{n} is enough—if a number has a factor greater than \sqrt{n} , it must also have a corresponding factor less than \sqrt{n} .

The screenshot shows the AI Assistant Coding interface with the completed Python script. The terminal at the bottom shows the script being run and a series of test examples. The AI summary remains the same, emphasizing the efficiency of the optimized logic.

```

88     if is_prime:
89         print(f"{num} is Prime")
90     else:
91         print(f"{num} is Composite")
92
93 # Test examples
94 print("\nExamples:")
95 print("0 → Neither")
96 print("1 → Neither")
97 print("2 → Prime")
98 print("17 → Prime")
99 print("20 → Composite")
100 print("-5 → Neither")

```

Key features:

- Edge cases handled: negatives, 0, 1 → classified as "Neither"
- Special case: 2 is Prime (only even prime)
- Optimized logic: Only checks factors up to \sqrt{n} using `while i * i <= num`
- Input validation: Classifies any number correctly as Prime, Composite, or Neither

This approach is efficient because checking up to \sqrt{n} is enough—if a number has a factor greater than \sqrt{n} , it must also have a corresponding factor less than \sqrt{n} .

Question 5: Zero-Shot Prompting (Perfect Number Check)

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

- Record the AI-generated code.
- Test the program with multiple inputs.
- Identify any missing conditions or inefficiencies in the logic.

PROMPT : check whether the number is perfect number or not in python.

The screenshot shows a software interface for AI-assisted coding. On the left, there's a code editor with a file named 'Assignment_3.1.py'. The code is as follows:

```
104 # Perfect number: equals the sum of its proper divisors (divisors except itself)
105
106 print("\n--- Perfect Number Check ---")
107
108 num = int(input("Enter a number: "))
109
110 if num <= 0:
111     print(f"{num} is Not a Perfect Number (must be positive)")
112 else:
113     # Find sum of all divisors except the number itself
114     sum_of_divisors = 0
115
116     for i in range(1, num):
117         if num % i == 0:
118             sum_of_divisors += i
119
120     # Check if perfect number
121     if sum_of_divisors == num:
122         print(f"{num} is A Perfect Number")
123         print(f"Divisors of {num}: 1 to {num-1}, Sum = {sum_of_divisors}")
124     else:
125         print(f"{num} is Not a Perfect Number")
126         print(f"Sum of divisors: {sum_of_divisors}")
127
128 # Examples
129 print("\nExamples:")
130 print("6 is Perfect (1 + 2 + 3 = 6)")
131 print("28 is Perfect (1 + 2 + 4 + 7 + 14 = 28)") Keep Undo ⌛ | 1 of 1 ↑ ↓
132 print("10 is Not Perfect (1 + 2 + 5 = 8)")


```

On the right side, there's a sidebar titled "PALINDROME CHECK FOR A NUMBER IN PYTHON". It contains the following text:

check whether the number is perfect number or not in python

- ✓ Read Python file
- ✓ Edited Assignment_3.1.py +33 -0

Done! Added TASK - V to check for perfect numbers.

How it works:

- A perfect number equals the sum of its proper divisors (all divisors except itself)
- Finds all divisors from 1 to n-1
- Sums them and compares with the original number
- Handles edge cases: negative numbers and 0

Examples:

- 6 is Perfect (1 + 2 + 3 = 6)
- 28 is Perfect (1 + 2 + 4 + 7 + 14 = 28)
- 10 is Not Perfect (1 + 2 + 5 = 8)

1 file changed +33 -0

Keep Undo ⌛

Describe what to build next

Agent Auto ENG IN 08:14 PM 12-01-2026

The screenshot shows a code editor with a Python file named `Assignment_3.1.py`. The code checks if a number is perfect, printing its divisors and their sum. It also includes examples for numbers 6, 28, and 10.

```

117     if num % i == 0:
118         sum_of_divisors += i
119
120     # Check if perfect number
121     if sum_of_divisors == num:
122         print(f"{num} is a Perfect Number")
123         print(f"Divisors of {num}: 1 to {num-1}, Sum = {sum_of_divisors}")
124     else:
125         print(f"{num} is Not a Perfect Number")
126         print(f"Sum of divisors: {sum_of_divisors}")
127
128 # Examples:
129 print("\nExamples:")
130 print("6 is Perfect (1 + 2 + 3 = 6)")
131 print("28 is Perfect (1 + 2 + 4 + 7 + 14 = 28)")
132 print("10 is Not Perfect (1 + 2 + 5 = 8)")
133

```

The AI Assistant Coding interface shows a chat window with the prompt "check whether the number is perfect number or not in python". Below it, a summary says "Done! Added TASK - V to check for perfect numbers." A "How it works:" section lists steps: finding divisors from 1 to n-1, summing them, and comparing to the original number. An "Examples:" section shows outputs for 6, 28, and 10.

Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even

Task:

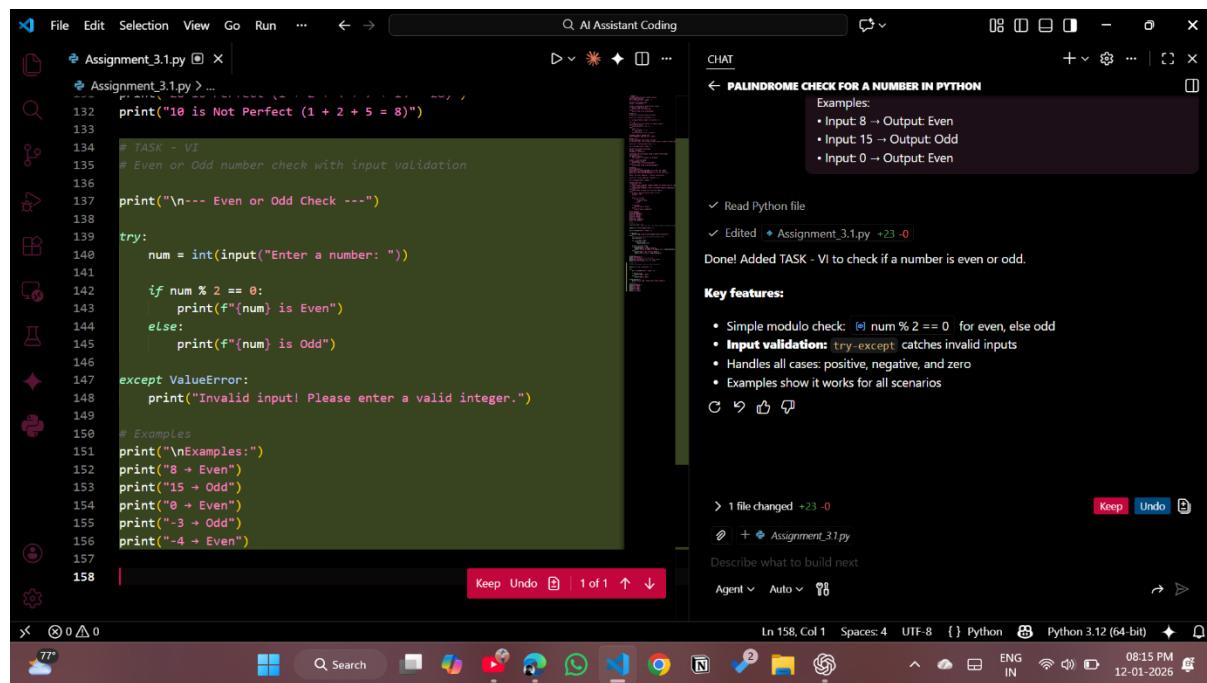
- Analyze how examples improve input handling and output clarity.

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

PROMPT : Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
 - Input: 15 → Output: Odd
 - Input: 0 → Output: Even



The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** Q AI Assistant Coding
- Code Area:** Assignment_3.1.py (Assignment_3.1.py > ...)
- Code Content:**

```
135 # Even or Odd number check with input validation
136
137 print("\n--- Even or Odd Check ---")
138
139 try:
140     num = int(input("Enter a number: "))
141
142     if num % 2 == 0:
143         print(f"{num} is Even")
144     else:
145         print(f"{num} is Odd")
146
147 except ValueError:
148     print("Invalid input! Please enter a valid integer.")
```
- Terminal:** PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding> & C:\Users\rishi\ppData\Local\Programs\Python\Python312\python.exe "c:/Users/rishi/OneDrive/Desktop/UNI/SEM - VI/AI Assistant Coding/Assignment_3.1.py"
--- Even or Odd Check ---
Enter a number: 34
34 is Even
- Examples:**

```
8 → Even
15 → Odd
0 → Even
-3 → Odd
-4 → Even
```
- Output:** PS C:\Users\rishi\OneDrive\Desktop\UNI\SEM - VI\AI Assistant Coding>
- AI Assistant Coding Panel:**
 - CHAT: PALINDROME CHECK FOR A NUMBER IN PYTHON
 - DESCRIPTION: Determine whether a given number is even or odd, including input validation.
 - EXAMPLES:
 - Input: 8 → Output: Even
 - Input: 15 → Output: Odd
 - Input: 0 → Output: Even
 - KEY FEATURES:
 - Simple modulo check: `if num % 2 == 0` for even, else odd
 - Input validation:** `try-except` catches invalid inputs
 - Handles all cases: positive, negative, and zero
 - Examples show it works for all scenarios
 - STATUS: ✓ Read Python file
✓ Edited Assignment_3.1.py +23 -0
 - MESSAGES: Done! Added TASK - VI to check if a number is even or odd.
- System Status:** Ln 135, Col 30, Spaces: 4, UTF-8, CRLF, Python 3.12 (64-bit), 08:16 PM, 12-01-2026