

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.

CODE:

```
A3.py > is_palindrome_number
1  # prompt: Write a Python function named is_palindrome_number(n)
2  # that returns True if the number is a palindrome and False otherwise.
3  # The function should handle negative numbers and zero properly.
4  # Do not use string reversal.
5  def is_palindrome_number(n):
6      if n < 0:
7          return False
8      if n == 0:
9          return True
10
11     original_number = n
12     reversed_number = 0
13
14     while n > 0:
15         digit = n % 10
16         reversed_number = reversed_number * 10 + digit
17         n //= 10
18
19     return original_number == reversed_number
20 while True:
21     num = int(input("Enter a number: "))
22
23     if is_palindrome_number(num):
24         print("Palindrome number")
25     else:
26         print("Not a palindrome number")
27
28     another = input("\nDo you want to check another number? (yes/no): ").strip().lower()
29
30     if another != 'yes':
31         break
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Enter a number: 121
Palindrome number

Do you want to check another number? (yes/no): yes
Enter a number: 103
Not a palindrome number

Do you want to check another number? (yes/no): no
```

ANALYSIS:

Q1. Palindrome Number (Zero-Shot)

- The AI generated correct logic without any examples.
- It handles negative numbers and zero properly.
- It uses math to reverse the number, not strings.
- The program works correctly for all tested inputs.

Conclusion: Zero-shot prompt was enough for this simple problem.

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.

CODE:

```
32 # prompt: Write a Python function named factorial(n)
33 # that returns the factorial of a given non-negative integer n.
34 # For example, Input: 5 → Output: 120
35 def factorial(n):
36     if n < 0:
37         raise ValueError("Factorial is not defined for negative numbers.")
38     if n == 0 or n == 1:
39         return 1
40     result = 1
41     for i in range(2, n + 1):
42         result *= i
43     return result
44 while True:
45     try:
46         num = int(input("Enter a non-negative integer: "))
47         if num < 0:
48             print("Error: Please enter a non-negative integer.")
49             continue
50         print(f"Factorial of {num} is {factorial(num)}")
51     except ValueError:
52         print("Error: Invalid input. Please enter a valid non-negative integer.")
53
54     another = input("\nDo you want to calculate another factorial? (yes/no): ").strip().lower()
55
56     if another != 'yes':
57         break
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Enter a non-negative integer: 5
Factorial of 5 is 120

Do you want to calculate another factorial? (yes/no): yes
Enter a non-negative integer: 10
Factorial of 10 is 3628800

Do you want to calculate another factorial? (yes/no): no
```

ANALYSIS:

Q2. Factorial (One-Shot)

- The example helped AI understand the expected output.
- The code handles 0, positive numbers, and gives error for negatives.
- Logic is clear and correct.

- **Conclusion:** One example improved accuracy and input handling.

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.

CODE:

```
A3.py > is_palindrome_number
58 # prompt: Write a Python function named is_armstrong_number(n)
59 # that returns True if the number is an Armstrong number and False otherwise.
60 # For example:
61 # Input: 153 → Output: Armstrong Number
62 # Input: 370 → Output: Armstrong Number
63 # Input: 123 → Output: Not an Armstrong Number
64 def is_armstrong_number(n):
65     if n < 0:
66         return False
67
68     num_str = str(n)
69     num_digits = len(num_str)
70     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
71
72     return sum_of_powers == n
73 while True:
74     try:
75         num = int(input("Enter a non-negative integer: "))
76         if num < 0:
77             print("Error: Please enter a non-negative integer.")
78             continue
79         if is_armstrong_number(num):
80             print("Armstrong Number")
81         else:
82             print("Not an Armstrong Number")
83     except ValueError:
84         print("Error: Invalid input. Please enter a valid non-negative integer.")
85
86     another = input("\nDo you want to check another number? (yes/no): ").strip().lower()
87
88     if another != 'yes':
89         break
```

Ln 8, Col 15 Spaces: 4 UTF-

OUTPUT:

```
Enter a non-negative integer: 153
Armstrong Number

Do you want to check another number? (yes/no): yes
Enter a non-negative integer: 100
Not an Armstrong Number

Do you want to check another number? (yes/no): no
```

ANALYSIS:

Q3. Armstrong Number (Few-Shot)

- Multiple examples helped AI understand the pattern clearly.
- The function correctly checks Armstrong numbers.
- It handles invalid and negative inputs.

Conclusion: Few-shot prompting improved correctness and clarity

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.

CODE:

```
90 # prompt: Write a Python function named classify_number(n)
91 # that classifies a given integer n as 'Prime', 'Composite', or 'Neither'.
92 # The function should handle input validation and optimize for efficiency.
93 def classify_number(n):
94     if n < 2:
95         return "Neither"
96     if n == 2:
97         return "Prime"
98     if n % 2 == 0:
99         return "Composite"
100
101     for i in range(3, int(n**0.5) + 1, 2):
102         if n % i == 0:
103             return "Composite"
104     return "Prime"
105 while True:
106     try:
107         num = int(input("Enter an integer: "))
108         classification = classify_number(num)
109         print(f"The number {num} is classified as: {classification}")
110     except ValueError:
111         print("Error: Invalid input. Please enter a valid integer.")
112
113     another = input("\nDo you want to classify another number? (yes/no): ").strip().lower()
114
115     if another != 'yes':
116         break
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Enter an integer: 2
The number 2 is classified as: Prime

Do you want to classify another number? (yes/no): yes
Enter an integer: 4
The number 4 is classified as: Composite

Do you want to classify another number? (yes/no): yes
Enter an integer: 0
The number 0 is classified as: Neither

Do you want to classify another number? (yes/no): no
```

ANALYSIS:

Q4. Prime / Composite / Neither (Context-Managed)

- The program is efficient using \sqrt{n} logic.
- It properly classifies all numbers including 0 and 1.
- Input validation is handled.

Conclusion: Giving clear instructions produced optimized code.

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.

CODE:

```
117 # prompt: Write a Python function named is_perfect_number(n)
118 # that returns True if the number is a perfect number and False otherwise.
119 def is_perfect_number(n):
120     if n <= 0:
121         return False
122
123     sum_of_divisors = 0
124     for i in range(1, n):
125         if n % i == 0:
126             sum_of_divisors += i
127
128     return sum_of_divisors == n
129 while True:
130     try:
131         num = int(input("Enter a positive integer: "))
132         if num <= 0:
133             print("Error: Please enter a positive integer.")
134             continue
135         if is_perfect_number(num):
136             print("Perfect Number")
137         else:
138             print("Not a Perfect Number")
139     except ValueError:
140         print("Error: Invalid input. Please enter a valid positive integer.")
141
142     another = input("\nDo you want to check another number? (yes/no): ").strip().lower()
143
144     if another != 'yes':
145         break
```


OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Enter a positive integer: 123
Not a Perfect Number

Do you want to check another number? (yes/no): yes
Enter a positive integer: 6
Perfect Number

Do you want to check another number? (yes/no): yes
Enter a positive integer: 8
Not a Perfect Number

Do you want to check another number? (yes/no): no
```

ANALYSIS:

Q5. Perfect Number (Zero-Shot)

- The logic is correct but checks all numbers till $n-1$ (slower).
- Works correctly for valid inputs.

Conclusion: Correct result, but not optimized.

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.

CODE:

```
A3.py > is_palindrome_number
146 # prompt: Write a Python function named classify_even_odd(n)
147 # that returns 'Even' if the number is even and 'Odd' if the number is odd.
148 # The function should handle input validation.
149 def classify_even_odd(n):
150     if not isinstance(n, int):
151         raise ValueError("Input must be an integer.")
152     return "Even" if n % 2 == 0 else "Odd"
153 while True:
154     user_input = input("Enter an integer: ")
155     try:
156         num = int(user_input)
157         classification = classify_even_odd(num)
158         print(f"The number {num} is classified as: {classification}")
159     except ValueError:
160         print("Error: Invalid input. Please enter a valid integer.")
161
162     another = input("\nDo you want to classify another number? (yes/no): ").strip().lower()
163
164     if another != 'yes':
165         break
166
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Do you want to check another number? (yes/no): no
Enter an integer: 10
The number 10 is classified as: Even

Do you want to classify another number? (yes/no): yes
Enter an integer: 3
The number 3 is classified as: Odd

Do you want to classify another number? (yes/no): yes
Enter an integer: 0
The number 0 is classified as: Even

Do you want to classify another number? (yes/no): no
```

ANALYSIS:

Q6. Even / Odd (Few-Shot)

- Examples helped generate clean logic.
- Handles invalid inputs safely.
- Output is clear and accurate.

Conclusion: Few-shot prompting improved reliability.