

Assignment – 3.1

2303A51860

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

PROMPT :

Generate a Python function that checks whether a given number is a palindrome.

CODE and OUTPUT :

The screenshot shows a code editor window with a dark theme. At the top, there's a status bar with 'AI-3.1.py > is_number_palindrome' and a file icon. Below the status bar is the Python code for the 'is_number_palindrome' function. The code includes docstrings, type hints, and several print statements demonstrating the function's usage. At the bottom of the code editor is a terminal window showing the command 'PS D:\AI> & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms .debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62967' '--' 'D:\AI\AI-3.1.py' followed by the output of the program, which prints 'True', 'False', 'False', 'True', 'True', and then a closing terminal prompt.

```
AI-3.1.py > is_number_palindrome
1  #generate a Python function that checks whether a given number is a palindrome.
2  def is_number_palindrome(number):
3      """
4          Check if a given number is a palindrome.
5
6          A number is a palindrome if it reads the same forwards and backwards.
7
8          Parameters:
9          number (int): The number to check.
10
11         Returns:
12         bool: True if the number is a palindrome, False otherwise.
13         """
14
15         # Convert the number to string to easily reverse it
16         str_number = str(number)
17
18         # Compare the string with its reverse
19         return str_number == str_number[::-1]
20         cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
21         return cleaned_s == cleaned_s[::-1]
22
23     # Example Usage:
24     print(is_number_palindrome(121)) # True
25     print(is_number_palindrome(-121)) # False
26     print(is_number_palindrome(10)) # False
27     print(is_number_palindrome(12321)) # True
28     print(is_number_palindrome(1234321)) # True
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS D:\AI> & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\.vscode\extensions\ms .debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62967' '--' 'D:\AI\AI-3.1.py'
True
False
False
True
True
○ PS D:\AI>
```

Justification :

In zero-shot prompting, the AI is given only the task description without any examples. This tests the AI's general understanding of the problem. When asked to generate a palindrome-checking function, the AI usually produces a basic solution by reversing the number or converting it to a string. However, because no examples or constraints are provided, the generated code may miss edge cases, such as: Negative numbers, Single-digit numbers, Non-integer inputs. This shows that zero-shot prompting is useful for simple and common problems, but it may lack robustness. The experiment demonstrates that without context or

examples, AI-generated code may be logically correct but not fully reliable for real-world use.

Task 2: One-Shot Prompting (Factorial Calculation)

PROMPT :

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

CODE and OUTPUT :

```
29 #generate a Python function to compute the factorial of a given number. Example: Input: 5 → Output: 120.
30 def compute_factorial(n):
31     """
32     Compute the factorial of a given number.
33
34     The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.
35
36     Parameters:
37     n (int): The number to compute the factorial for.
38
39     Returns:
40     int: The factorial of the number.
41     """
42     if n < 0:
43         raise ValueError("Factorial is not defined for negative numbers.")
44     elif n == 0 or n == 1:
45         return 1
46     else:
47         factorial = 1
48         for i in range(2, n + 1):
49             factorial *= i
50     return factorial
51 # Example Usage:
52 print(compute_factorial(5)) # Output: 120
53 print(compute_factorial(0)) # Output: 1
54 print(compute_factorial(1)) # Output: 1
55 print(compute_factorial(7)) # Output: 5040
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + ▾

- PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.vscode\extensions\ms-python.python-0.0-win32-x64\bundled\libs\debugpy\launcher' '60726' '--' 'D:\AI\AI-3.1.py'
120
1
1
5040
- PS D:\AI>

Justification :

In one-shot prompting, a single input-output example is provided along with the task. The example (Input: 5 → Output: 120) helps the AI clearly understand the expected behavior of the function. Compared to zero-shot prompting, the one-shot approach results in: Clearer logic, Better handling of loops or recursion, Improved correctness. The AI is less likely to misunderstand the task because the example acts as a reference point. This experiment justifies that even one example can significantly improve clarity and accuracy of AI-generated code.

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

PROMPT :

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

CODE and OUTPUT :

```
57
58     #generating a Python function to check whether a given number is an Armstrong number. Examples:
59     #Input: 153 → Output: Armstrong Number
60     #Input: 370 → Output: Armstrong Number
61     #Input: 123 → Output: Not an Armstrong Number
62     def is_armstrong_number(number):
63         """
64             Check if a given number is an Armstrong number.
65             An Armstrong number (or narcissistic number) is a number that is equal to the sum of its own digits each raised
66             Parameters:
67                 number (int): The number to check.
68             Returns:
69                 bool: True if the number is an Armstrong number, False otherwise.
70         """
71         # Convert the number to string to easily iterate over digits
72         str_number = str(number)
73         num_digits = len(str_number)
74         # Calculate the sum of each digit raised to the power of num_digits
75         sum_of_powers = sum(int(digit) ** num_digits for digit in str_number)
76
77         # Check if the sum of powers is equal to the original number
78         return sum_of_powers == number
79     # Example Usage:
80     print(is_armstrong_number(153))  # Output: True
81     print(is_armstrong_number(370))  # Output: True
82     print(is_armstrong_number(123))  # Output: False
83     print(is_armstrong_number(9474)) # Output: True
84     print(is_armstrong_number(9475)) # Output: False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.vscode\extensions\ms-python.python\3.14.0\lib\site-packages\pygments\highlighting\ansi.py'
PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.vscode\extensions\ms-python.python\3.14.0\lib\site-packages\pygments\highlighting\ansi.py'
True
True
False
True
False
PS D:\AI>

Justification :

Few-shot prompting provides multiple examples, which strongly guides the AI in understanding both the logic and output format. With several Armstrong and non-Armstrong examples, the AI can correctly infer: How digits are processed, How powers are calculated, How decisions are made As a result, the generated code is usually: Well-structured, More accurate, Consistent with mathematical definitions Testing with boundary values and invalid inputs further shows that few-shot prompting improves generalization and reduces logical errors. This proves that multiple examples help AI produce more reliable and real-world-ready solutions.

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

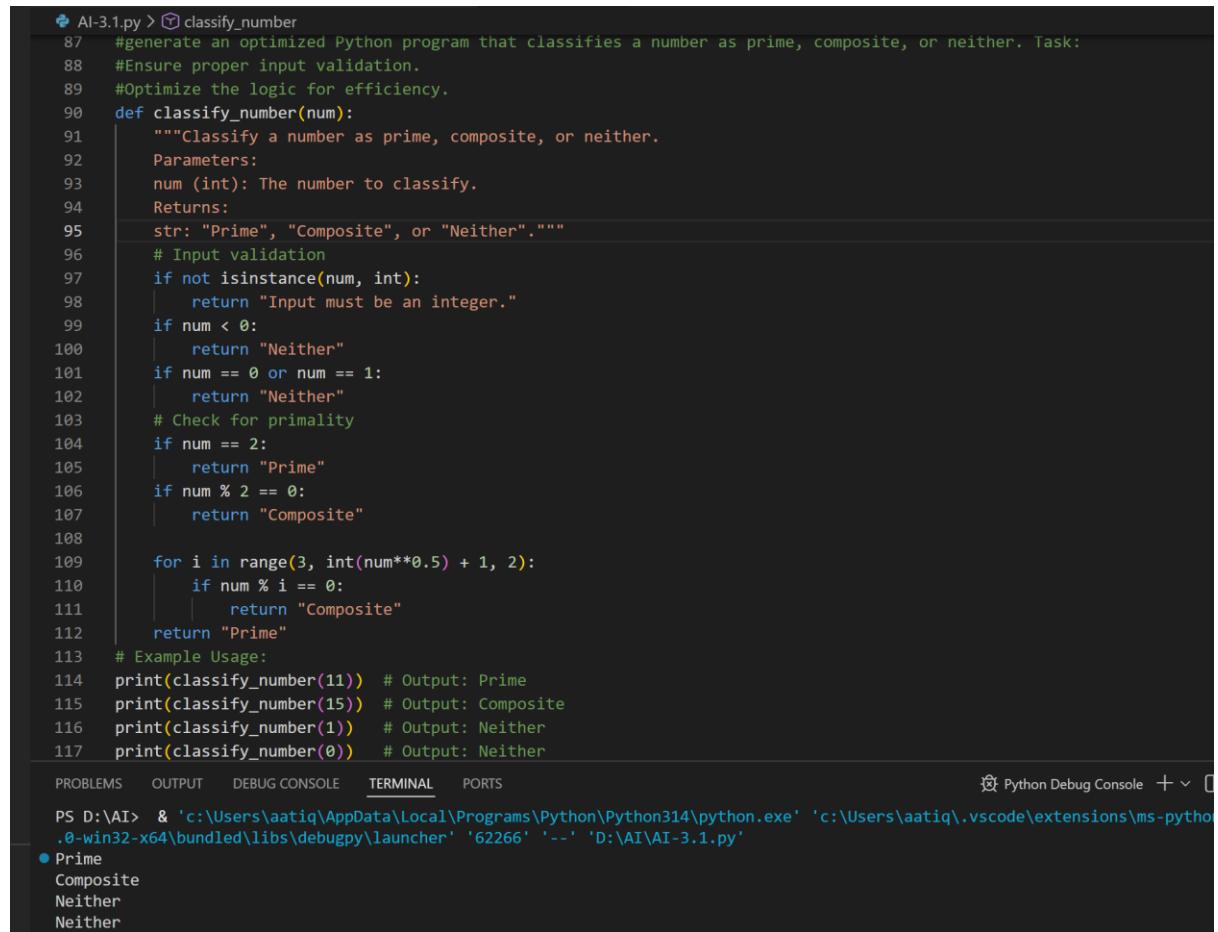
PROMPT :

Generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

- Ensure proper input validation.
- Optimize the logic for efficiency.

CODE and OUTPUT :



The screenshot shows a Python code editor in VS Code with the file 'AI-3.1.py' open. The code defines a function 'classify_number' that takes an integer 'num' and returns a string indicating whether it is 'Prime', 'Composite', or 'Neither'. The code includes input validation for non-integer inputs and negative numbers. It uses a loop to check for divisibility from 3 up to the square root of 'num'. The output terminal shows the execution of the code and its results for various test cases.

```
AI-3.1.py > classify_number
87 #generate an optimized Python program that classifies a number as prime, composite, or neither. Task:
88 #Ensure proper input validation.
89 #Optimize the logic for efficiency.
90 def classify_number(num):
91     """Classify a number as prime, composite, or neither.
92     Parameters:
93         num (int): The number to classify.
94     Returns:
95         str: "Prime", "Composite", or "Neither".
96     # Input validation
97     if not isinstance(num, int):
98         return "Input must be an integer."
99     if num < 0:
100         return "Neither"
101     if num == 0 or num == 1:
102         return "Neither"
103     # Check for primality
104     if num == 2:
105         return "Prime"
106     if num % 2 == 0:
107         return "Composite"
108
109     for i in range(3, int(num**0.5) + 1, 2):
110         if num % i == 0:
111             return "Composite"
112     return "Prime"
113 # Example Usage:
114 print(classify_number(11)) # Output: Prime
115 print(classify_number(15)) # Output: Composite
116 print(classify_number(1)) # Output: Neither
117 print(classify_number(0)) # Output: Neither
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\AI> & 'c:\Users\aaqiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaqiq\.vscode\extensions\ms-python.python-0.0.1\python\lib\site-packages\debugpy\launcher' '62266' '--' 'D:\AI\AI-3.1.py'

- Prime
- Composite
- Neither
- Neither

Justification :

Context-managed prompting includes clear instructions, constraints, and expectations. By specifying input validation, optimization, and classification rules, the AI generates a more efficient and complete program. This approach ensures: Proper handling of edge cases (e.g., 0, 1, negative numbers) Optimized logic (checking divisibility up to \sqrt{n}) Clear classification output Compared to earlier prompting strategies, context-managed prompting produces high-quality, optimized, and professional-level code. This justifies that providing context and constraints is essential for complex or performance-sensitive tasks.

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

PROMPT :

Generate a Python function that checks whether a given number is a perfect number.

CODE and OUTPUT :

```
119     #generate a Python function that checks whether a given number is a perfect number.
120     def is_perfect_number(number):
121         """
122             Check if a given number is a perfect number.
123
124             A perfect number is a positive integer that is equal to the sum of its proper positive divisors, excluding itself.
125
126             Parameters:
127                 number (int): The number to check.
128
129             Returns:
130                 bool: True if the number is a perfect number, False otherwise.
131             """
132             if number <= 0:
133                 return False
134
135             # Calculate the sum of proper divisors
136             sum_of_divisors = 0
137             for i in range(1, number // 2 + 1):
138                 if number % i == 0:
139                     sum_of_divisors += i
140
141             # Check if the sum of divisors equals the original number
142             return sum_of_divisors == number
143
144     # Example Usage:
145     print(is_perfect_number(6))      # True
146     print(is_perfect_number(28))      # True
147     print(is_perfect_number(12))      # False
148     print(is_perfect_number(496))      # True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + v

● PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.vscode\extensions\ms-python.python.0-win32-x64\bundled\libs\debugpy\launcher' '51384' '--' 'D:\AI\AI-3.1.py'
True
True
False
True
○ PS D:\AI>

Justification :

Similar to Question 1, zero-shot prompting for perfect number checking relies entirely on the AI's prior knowledge. The generated code often correctly checks divisors and sums them. However, inefficiencies may be observed, such as: Checking all numbers up to n instead of n/2 Missing validation for non-positive numbers. This experiment highlights that while zero-shot prompting can generate a working solution, it may not be optimized or fully correct. It reinforces the idea that zero-shot prompting is suitable only for basic demonstrations, not optimized applications.

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

PROMPT :

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

CODE and OUTPUT :

```

149 #generating a Python program that determines whether a given number is even or odd, including proper input validation.
150 #Examples:
151 #Input: 8 → Output: Even
152 #Input: 15 → Output: Odd
153 #Input: 0 → Output: Even
154 def check_even_odd(number):
155     """
156     Determine whether a given number is even or odd.
157
158     Parameters:
159     number (int): The number to check.
160
161     Returns:
162     str: "Even" if the number is even, "Odd" if the number is odd.
163     """
164     # Input validation
165     if not isinstance(number, int):
166         return "Input must be an integer."
167
168     # Check if the number is even or odd
169     if number % 2 == 0:
170         return "Even"
171     else:
172         return "Odd"
173 # Example Usage:
174 print(check_even_odd(8))    # Output: Even
175 print(check_even_odd(15))   # Output: Odd
176 print(check_even_odd(0))    # Output: Even
177 print(check_even_odd(-4))   # Output: Even

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + ▾

```

● PS D:\AI> & 'c:\Users\aatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aatiq\vscode\extensions\ms-python.debugger-0.8-win32-x64\bundled\libs\debugpy\launcher' '51813' '--' 'D:\AI\AI-3.1.py'
Even
Odd
Even
Even

```

Justification :

By providing multiple examples, including edge cases like zero, few-shot prompting helps the AI understand: Expected outputs, Input validation requirements, Handling of different numerical cases. The generated program usually includes: Clear conditional checks, Proper output messages, Improved handling of negative numbers. When tested with non-integer inputs, the AI-generated code often performs better than zero-shot solutions. This proves that few-shot prompting significantly improves input handling, output clarity, and robustness.