

Assignment – 3.1

2303A51T01

B.GNANESHWAR

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 tabs for PROBLEMS (2), OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. Below the status bar is the code for 'AI-3.1.py'. The code defines a function 'is_number_palindrome' that checks if a given number is a palindrome by comparing it with its reverse. It includes docstrings and several print statements demonstrating the function's usage. At the bottom of the code editor is a terminal window showing the execution of the script and its output. The terminal output shows the script being run from a directory 'D:\AI', and it prints five lines of boolean values: True, False, False, True, and True. The terminal window has a status bar at the bottom with tabs for PS, D:, and AI.

```
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
21     cleaned_s = ''.join(char.lower() for char in s if char.isalnum())
22     return cleaned_s == cleaned_s[::-1]
23
24     # Example Usage:
25     print(is_number_palindrome(121)) # True
26     print(is_number_palindrome(-121)) # False
27     print(is_number_palindrome(10)) # False
28     print(is_number_palindrome(12321)) # True
29     print(is_number_palindrome(1234321)) # True
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.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>
```

OBSERVATION :-

In zero-shot prompting, the model receives only the problem statement without any supporting examples. This checks the AI's basic understanding of the concept. While generating a palindrome-checking function, the AI typically uses simple logic such as reversing the number or converting it into a string. However, due to the lack of constraints or examples, the solution may ignore important edge cases like negative values, single-digit numbers, or

invalid inputs. This indicates that although zero-shot prompting works well for simple tasks, it may not produce fully robust or real-world-ready code without additional guidance.

Task 2: One-Shot Prompting:-

PROMPT:

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

CODE and OUTPUT:

The screenshot shows a VS Code interface with a terminal window open. The terminal displays the generated Python code for a factorial function and its execution. The code is as follows:

```
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
```

Below the code, the terminal shows the execution of the function with various inputs and their outputs. The terminal tabs at the bottom include PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The status bar at the bottom right shows "Python Debug Console".

```
● PS D:\AI> & 'c:\Users\aaqiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaqiq\.vscode\extensions\ms-python.python-0.0.1\lib\site-packages\debugpy\launcher' '60726' '--' 'D:\AI\AI-3.1.py'
120
1
1
5040
○ PS D:\AI>
```

OBSERVATION :

One-shot prompting provides the AI with one example along with the task description. The given example (Input: 5 → Output: 120) helps the model clearly understand the expected result and logic. Compared to zero-shot prompting, this approach leads to more accurate implementations with correct looping or recursion. The presence of a single example reduces ambiguity and improves correctness, showing that even minimal guidance can significantly enhance the quality 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
80     # Example Usage:
81     print(is_armstrong_number(153)) # Output: True
82     print(is_armstrong_number(370)) # Output: True
83     print(is_armstrong_number(123)) # Output: False
84     print(is_armstrong_number(9474)) # Output: True
85     print(is_armstrong_number(9475)) # Output: False
```

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\3.14.0\lib\site-packages\IPython\utils\py3compat.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\IPython\utils\py3compat.py'
True
True
False
True
True
False
PS D:\AI>
```

OBSERVATION :

Few-shot prompting includes multiple examples, which strongly influences how the AI interprets the problem. By seeing both Armstrong and non-Armstrong cases, the model better understands digit extraction, power calculations, and decision-making logic. This results in cleaner, more structured, and mathematically accurate code. Additionally, the AI performs better when handling different test cases. Overall, providing several examples improves reliability and reduces logical mistakes, making the solution more suitable for real-world use.

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 file named AI-3.1.py in the VS Code editor. The code defines a function classify_number that takes an integer num and returns "Prime", "Composite", or "Neither". It includes input validation, handles edge cases (0, 1), and uses a square root check for primality up to \sqrt{n} . The code is annotated with docstrings and comments. The terminal below shows the execution of the script with example inputs and their outputs.

```
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
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 Python Debug Console + D

```
PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.vscode\extensions\ms-python
.0-win32-x64\bundled\libs\debugpy\launcher' '62266' '--' 'D:\AI\AI-3.1.py'
● Prime
Composite
Neither
Neither
```

OBSERVATION :

Context-managed prompting supplies detailed instructions, constraints, and performance expectations. By specifying input validation, efficiency requirements, and classification rules, the AI produces a well-optimized and complete program. The generated solution correctly handles special cases such as 0, 1, and negative numbers, and applies efficient logic by checking divisibility only up to \sqrt{n} . Compared to simpler prompting techniques, this method results in professional-quality, optimized, and reliable code, highlighting the importance of clear context for complex 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 + ⌂

PS D:\AI> & 'c:\Users\aaatique\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatique\.vscode\extensions\ms-python.python_0.1.1\lib\site-packages\debugpy\launcher' '51384' '--' 'D:\AI\AI-3.1.py'

True
True
False
True
True

PS D:\AI>

OBSERVATION :

In this zero-shot scenario, the AI depends entirely on its prior knowledge to generate the solution. The code usually identifies divisors and calculates their sum correctly. However, it may include inefficiencies such as checking all numbers up to n or failing to validate non-positive inputs. This demonstrates that while zero-shot prompting can generate functional code, it often lacks optimization and thorough validation. Therefore, it is better suited for basic demonstrations rather than efficient or production-level solutions.

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
● PS D:\AI> & 'c:\Users\aaatiq\AppData\Local\Programs\Python\Python314\python.exe' 'c:\Users\aaatiq\.vscode\extensions\ms-python.python.0.1.0\lib\site-packages\debugpy\launcher' '51813' '--' 'D:\AI\AI-3.1.py'
Even
Odd
Even
Even

```

OBSERVATION :

Few-shot prompting, supported by multiple examples including edge cases like zero, helps the AI clearly understand expected behavior and output formatting. The generated program typically includes proper conditional checks, clear messages, and better handling of negative numbers. Compared to zero-shot approaches, the solution also performs better when dealing with invalid or unexpected inputs. This confirms that providing multiple examples improves robustness, accuracy, and clarity in AI-generated programs.