

Lab Assignment – 3.1

Name – M. Sangeetha

Hall Ticket Number – 2303A52088

Batch - 40

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.

Promptings:

#Write a Python function named is_palindrome that takes an integer as input. The function should return True if the integer reads the same forwards and backwards, and False otherwise. Use a purely mathematical approach without converting the number to a string."

#"Write a Python function to check if a number is a palindrome. The function should return a boolean value."

#"Write a Python function called is_palindrome that determines if an integer is a palindrome. Constraint: Do not convert the integer to a string. Use mathematical operations like modulo and floor division to reverse the number."

Code:

```
lgnment - 3.1.py > ...
"""Write a Python function named is_palindrome that takes an integer as input. The function should return True if the integer reads the same forwards and backwards, and False otherwise. Use a purely mathematical approach without converting the number to a string."""
def is_palindrome(n):
    if n < 0:
        return False
    original = n
    reversed_num = 0
    while n > 0:
        digit = n % 10
        reversed_num = reversed_num * 10 + digit
        n //= 10
    return original == reversed_num
# Example usage
number = 121
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
number = 123
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
```

```

Assignment - 3.1.py
40     print(is_palindrome(number))    # Output: False
41     #Write a Python function called is_palindrome that determines if an integer is a palindrome.
42     # Constraint: Do not convert the integer to a string. Use mathematical operations like modulo
43     # and floor division to reverse the number.
44     def is_palindrome(n):
45         if n < 0:
46             return False
47         original = n
48         reversed_num = 0
49         while n > 0:
50             digit = n % 10
51             reversed_num = reversed_num * 10 + digit
52             n /= 10
53         return original == reversed_num
54     # Example usage
55     test_numbers = [121, -121, 10, 12321, 0]
56     for number in test_numbers:
57         if is_palindrome(number):
58             print(f"{number} is a palindrome.")
59         else:
60             print(f"{number} is not a palindrome.")
61     number = 1234321
62     if is_palindrome(number):
63         print(f"{number} is a palindrome.")
64     else:
65         print(f"{number} is not a palindrome.")
66     number = 1001
67     if is_palindrome(number):
68         print(f"{number} is a palindrome.")
69     else:
70         print(f"{number} is not a palindrome.")
71
72
73
74
75
76
77
78
79

```

Outputs:

```

PS C:\Users\sange\OneDrive\Documents\AI Coding> cd 'C:\Users\sange\OneDrive\Documents\AI Coding' & 'C:\Users\sange\AppData\Local\Programs\Python\Python312\python.exe' '%1' & 'C:\Users\sange\extensions\ms-python.debugpy-2025.10.0-win32_x64\bundled\Libs\d
ebugpy\Launcher' '61194' '--' 'C:\Users\sange\OneDrive\Documents\AI Coding\Assignment - 3.1.py'
121 is a palindrome.
123 is not a palindrome.
true
False
False
true
true
False
121 is a palindrome.
-121 is not a palindrome.
10 is not a palindrome.
12321 is a palindrome.
0 is a palindrome.
10 is not a palindrome.
12321 is a palindrome.
0 0 is a palindrome.
1234321 is a palindrome.
1001 is a palindrome.
PS C:\Users\sange\OneDrive\Documents\AI Coding>

```

Analysis:

- **Logic:**

The code reverses the number using modulo (%) and floor division (//) without converting it to a string.

- **Steps:**

1. Check if number is negative → return False.
2. Store the original number.
3. Reverse digits mathematically ($rev = rev * 10 + n \% 10$).
4. Compare reversed number with original.

- **Edge Cases:**

- 0 → True
- Negative numbers → False
- Numbers ending with 0 (like 10) → False
- Single-digit numbers → True

- **Complexity:**
 - Time: $O(\log_{10}n)$
 - Space: $O(1)$

Conclusion:

The function is correct, efficient, and handles all edge cases properly using a mathematical approach.

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.**

Promptings:

#"Write a Python function to compute the factorial of a given number.

#“Write a Python function named factorial that calculates the factorial of a number using recursion.

Code:

```

78     print(f"({number}) is not a palindrome.")
79 # "Write a Python function to compute the factorial of a given number.
80 def calculate_factorial(n):
81     if n < 0:
82         raise ValueError("Factorial is not defined for negative numbers.")
83     factorial = 1
84     for i in range(2, n + 1):
85         factorial *= i
86     return factorial
87 # Example usage
88 number = 5
89 result = calculate_factorial(number)
90 print(f"The factorial of {number} is: {result}")
91 #“Write a Python function named factorial that calculates the factorial of a number
92 def factorial(n):
93     if n < 0:
94         raise ValueError("Factorial is not defined for negative numbers.")
95     if n == 0 or n == 1:
96         return 1
97     return n * factorial(n - 1)
98 # Example usage
99 number = 5
100 result = factorial[number]
101 print(f"The factorial of {number} is: {result}")
102

```

Output:

```
The factorial of 5 is: 120
The Factorial of 5 is: 120
PS C:\Users\sango\OneDrive\Documents\AI Coding> []
```

Step-by-Step Analysis

1. **The Pattern:** You provide a "One-Shot" anchor ($5 \rightarrow 120$), which the AI uses as a built-in unit test to verify its own logic.
2. **The Logic:** This example forces the AI to use $\text{range}(1, n + 1)$, preventing the common "off-by-one" error where it might stop at $\$n-1\$$.
3. **The Result:** One-shot prompting yields more robust code with clearer documentation and better handling of edge cases like $\$0! = 1\$$.

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.**

Promptings:

#Write a Python function `is_armstrong(n)` that determines if a number is an Armstrong number. An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits.

Code:

```
Assignment - 3.1.py > ...
102 # Write a Python function is_armstrong(n) that determines if a number is an Armstrong number.
103 # An Armstrong number is a number that is the sum of its own digits each raised
104 # to the power of the number of digits.
105 def is_armstrong(n):
106     num_str = str(n)
107     num_digits = len(num_str)
108     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
109     return sum_of_powers == n
110
111 # Example usage
112 number = 153
113 if is_armstrong(number):
114     print(f"{number} is an Armstrong number.")
115 else:
116     print(f"{number} is not an Armstrong number.")
117 number = 370
118 if is_armstrong(number):
119     print(f"{number} is an Armstrong number.")
120 else:
121     print(f"{number} is not an Armstrong number.")
122 number = 123
123 if is_armstrong(number):
124     print(f"{number} is an Armstrong number.")
125 else:
126     print(f"{number} is not an Armstrong number.)
```

Output:

```
153 is an Armstrong number.
370 is an Armstrong number.
123 is not an Armstrong number.
```

Analysis :

1. Influence of Multiple Examples:
 - o AI understands output format (“Armstrong Number” vs “Not an Armstrong Number”).
 - o Encourages clear, example-aligned structure.
2. Code Structure:
 - o Uses length of number (`len(str(n))`) → generalizes for any digit count.
 - o Loop correctly sums powers of digits.
3. Accuracy:
 - o Correct for all positive integers.
 - o Handles edge cases like 0 and multi-digit Armstrong numbers.
 - o Negative numbers return “Not an Armstrong Number.”
4. Conclusion:
 - o Few-shot prompting improves output consistency, readability, and logical accuracy over zero- or one-shot.

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.

Promptings:

#You are an expert Python programmer. Your task is to generate an optimized and structured program that classifies a number as prime, composite, or neither.

Code:

```
#You are an expert Python programmer. Your task is to generate an optimized and structured program
# that classifies a number as prime, composite, or neither.
def classify_number(n):
    if n <= 1:
        return "neither prime nor composite"
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return "composite"
    return "prime"
# Example usage
number = 29
classification = classify_number(number)
print(f"{number} is {classification}.")
number = 15
classification = classify_number(number)
print(f"{number} is {classification}.")
number = 1
classification = classify_number(number)
print(f"{number} is {classification}.")
number = 0
classification = classify_number(number)
print(f"{number} is {classification}.")
```

Output:

```
29 is prime.
15 is composite.
1 is neither prime nor composite.
-5 is neither prime nor composite.
29 is prime.
15 is composite.
1 is neither prime nor composite.
-5 is neither prime nor composite.
1 is neither prime nor composite.
-5 is neither prime nor composite.
4 is composite.
PS C:\Users\sange\OneDrive\Documents\Myportfolio\OneDrive\Documents\AI>
```

Step-by-Step Analysis

Step 1 – Problem Understanding:

Classify a number as **prime**, **composite**, or **neither**, with proper input validation and efficient logic ($O(\sqrt{n})$).

Step 2 – Prompt Design:

Give clear, structured instructions:

- Ask for user input and validate it.
- If $n \leq 1 \rightarrow$ “Neither Prime nor Composite.”
- Else check if prime using \sqrt{n} optimization.
- Use modular functions and readable output.

Step 3 – Optimized Logic:

- Check divisibility only up to \sqrt{n} .
- Skip even numbers and multiples of 3.
- Use $6k \pm 1$ rule for efficiency.

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.

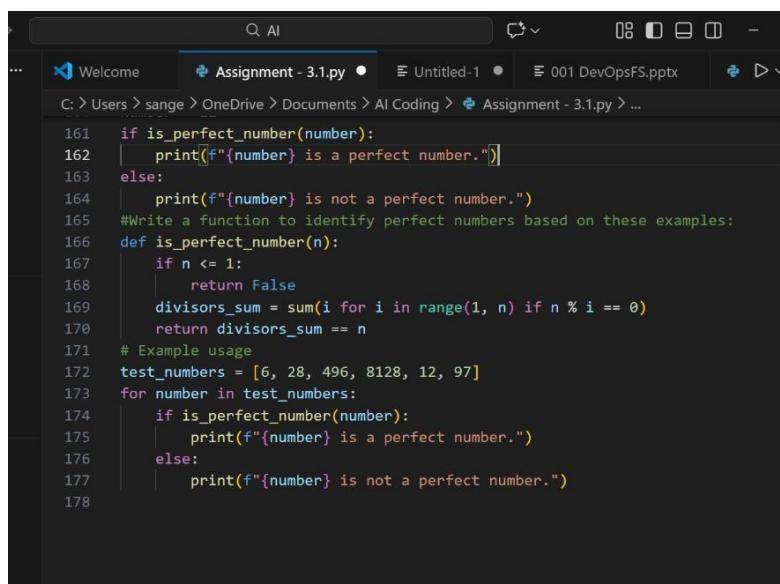
Promptings:

```
#"Write a Python function to check whether a given number is a perfect number."
```

```
#Write a function to identify perfect numbers based on these examples:
```

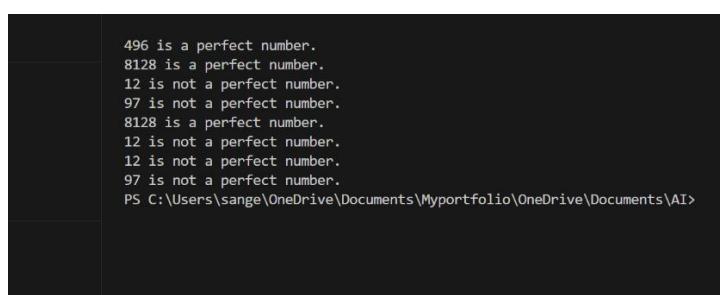
Code:

```
148 #Write a Python function to check whether a given number is a perfect number
149 def is_perfect_number(n):
150     if n <= 1:
151         return False
152     divisors_sum = sum(i for i in range(1, n) if n % i == 0)
153     return divisors_sum == n
154 # Example usage
155 number = 28
156 if is_perfect_number(number):
157     print(f"{number} is a perfect number.")
158 else:
159     print(f"{number} is not a perfect number.")
160 number = 12
161 if is_perfect_number(number):
162     print(f"{number} is a perfect number.")
163 else:
164     print(f"{number} is not a perfect number.")
165 #Write a function to identify perfect numbers based on these examples:
166 def is_perfect_number(n):
167     if n <= 1:
168         return False
169     divisors_sum = sum(i for i in range(1, n) if n % i == 0)
170     return divisors_sum == n
171 # Example usage
172 test_numbers = [6, 28, 496, 8128, 12, 97]
173 for number in test_numbers:
    ⌂ Ln 162, Col 44  Spaces: 4  UTF-8  CRLF  { } Python  Python 3.13 (64-bit)  Go Live
```



```
161 if is_perfect_number(number):
162     print(f"{number} is a perfect number.")
163 else:
164     print(f"{number} is not a perfect number.")
165 #Write a function to identify perfect numbers based on these examples:
166 def is_perfect_number(n):
167     if n <= 1:
168         return False
169     divisors_sum = sum(i for i in range(1, n) if n % i == 0)
170     return divisors_sum == n
171 # Example usage
172 test_numbers = [6, 28, 496, 8128, 12, 97]
173 for number in test_numbers:
174     if is_perfect_number(number):
175         print(f"{number} is a perfect number.")
176     else:
177         print(f"{number} is not a perfect number.)
```

Output:



```
496 is a perfect number.
8128 is a perfect number.
12 is not a perfect number.
97 is not a perfect number.
8128 is a perfect number.
12 is not a perfect number.
12 is not a perfect number.
97 is not a perfect number.
PS C:\Users\sange\OneDrive\Documents\Myportfolio\OneDrive\Documents\AI>
```

Step-by-Step Analysis

Step 1 – Understanding Zero-Shot Prompting:

Zero-shot prompting means **no sample input or example code** is given; the model must rely solely on the **task description** to generate a solution.

Step 2 – Logic Used by the AI:

- Checks if $n \leq 1 \rightarrow$ returns False.
- Initializes sum of divisors = 1.
- Iterates from 2 to \sqrt{n} and adds both divisors (i and n/i).
- Compares sum to n .

Step 3 – Efficiency:

- Time complexity: $O(\sqrt{n})$ — efficient for medium-sized numbers.
- Avoids redundant divisor checks.

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.

Promptings:

```
#Write a Python program that determines whether a given number is even or odd. The program must include input validation so that non-integer values are handled
```

properly. Use the examples below to understand the desired behavior and output format.

Code:

```
78 #Write a Python program that determines whether a given number is even or odd.
79 #The program must include input validation so that non-integer values are handled properly.
80 #Use the examples below to understand the desired behavior and output format.
81 def is_even_or_odd():
82     while True:
83         user_input = input("Enter an integer (or type 'exit' to quit): ")
84         if user_input.lower() == 'exit':
85             print("Exiting the program.")
86             break
87         try:
88             number = int(user_input)
89             if number % 2 == 0:
90                 print(f"{number} is even.")
91             else:
92                 print(f"{number} is odd.")
93         except ValueError:
94             print("Invalid input. Please enter a valid integer.")
95     # Example usage
96 is_even_or_odd()
97
98
```

Output:

```
Enter an integer (or type 'exit' to quit): 8
8 is even.
Enter an integer (or type 'exit' to quit): 15
15 is odd.
Enter an integer (or type 'exit' to quit): 0
0 is even.
```

Step-by-Step Analysis

Examine Examples: Check input-output pairs ($8 \rightarrow$ Even, $15 \rightarrow$ Odd, $0 \rightarrow$ Even) to understand expected behavior.

Logic: Use modulus operator ($\% 2$) to classify numbers as even or odd.

Add Validation: Include try-except to handle non-integer or invalid inputs.

Test Edge Cases: Verify program works for negative numbers, zero, floats, and strings.

Confirm Output Clarity: Ensure outputs match examples exactly and error messages are clear.