AI-ASSISTED CODING

ASSIGNMENT-3

Task-1:
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:

```
#generate a code wheather a given number is palindrome number or not
```

Code:

```
def is_palindrome(number):
    num_str = str(number)
    return num_str == num_str[::-1]
n=int(input("Enter a number to check if it's a palindrome: "))
print(is_palindrome(n))
```

Output:

```
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>  & 'c:\Users\nagas\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\nagas\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56208' '--' 'C:\Users\nagas\OneDrive\Desktop\AI Lab\Lab3.py'
Enter a number to check if it's a palindrome: 521
False
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

Analysis:
The AI can write correct code without examples, but it may not handle all edge cases. Zero-shot is simple but less careful for tricky inputs.

Task-2:
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:

```
#example:input:5 ->output:120
#generate a code to find factorial of a given number
```

Code:

```
def factorial(number):
    if number == 0 or number == 1:
        return 1
    else:
        return number * factorial(number - 1)
num=int(input("Enter a number to find its factorial: "))
print(f"The factorial of {num} is {factorial(num)}")
```

Output:

```
Enter a number to find its factorial: 34
The factorial of 34 is 295232799039604140847618609643520000000
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

Analysis:
Giving one example helps the AI understand what output is expected. The code is clearer and more likely to be correct than zero-shot.

Task-3:
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:

```
#input:153 -> output:Amstrong number
#input:123 -> output:not an Amstrong number
#input:370 -> output:Amstrong number
#check weather a given number is amstrong number or not
```

Code:

```
def is_armstrong(number):
```

```
    num_str = str(number)
    num_digits = len(num_str)
    i=0
    sum_of_powers=0
    while i < num_digits:
        digit = int(num_str[i])
        sum_of_powers += digit ** num_digits
        i += 1
    return sum_of_powers == number
input_number = int(input("Enter a number to check if it's an Armstrong
number: "))
if is_armstrong(input_number):
    print(f"{input_number} is an Armstrong number.")
else:
    print(f"{input_number} is not an Armstrong number.")
```

Output:

```
Enter a number to check if it's an Armstrong number: 76523
76523 is not an Armstrong number.
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

Analysis:
Providing multiple examples guides the AI to write accurate and structured code. It handles different cases better than zero- or one-shot.

Task-4:
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:

```
#generate code for a given number with given constraints
#prime,composite,or neither
```

Code:

```
def classify_number(number):
```

```
    if number <= 1:
        return "neither prime nor composite"
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return "composite"
    return "prime"
num = int(input("Enter a number to classify: "))
result = classify_number(num)
print(f"{num} is {result}.")
```

Output:

```
Enter a number to classify: 133
133 is composite.
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

Analysis:
Clear instructions and rules make the AI write faster and optimized code. It also checks inputs properly and avoids mistakes.

Task-5:
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 weather a given number is perfect or not
```

Code:

```
def is_perfect(number):
    if number < 1:
        return False
    i=1
    sum_of_divisors = 0
    while i < number:
        if number % i == 0:

            sum_of_divisors += i
        i += 1
    return sum_of_divisors == number
```

```
if is_perfect(123):
    print(f"{123} is a Perfect number.")
else:
    print(f"{123} is not a Perfect number.")
if is_perfect(6):
    print(f"{6} is a Perfect number.")
else:
    print(f"{6} is not a Perfect number.")
```

Output:

```
123 is not a Perfect number.
6 is a Perfect number.
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

Analysis:
AI can solve the problem without examples, but it may be slow or miss conditions. Zero-shot works but not for all edge cases.

Task-6:
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:

```
#input-8 -> output:Even
#input-7 -> output:Odd
#check weather a given number is odd or even
```

Code:

```
def is_even(number):
    return "Even" if number % 2 == 0 else "Odd"

print(is_even(165))
```

```
print(is_even(44))
print(is_even(0))
```

Output:

```
Odd
Even
Even
PS C:\Users\nagas\OneDrive\Desktop\AI Lab>
```

Analysis:
Giving multiple examples helps the AI clearly understand how to classify numbers as even or odd. It also improves input handling for negative numbers and makes the output clearer.