Name: Shryitha suddala

Hall ticket:2303A51637

Batch:29

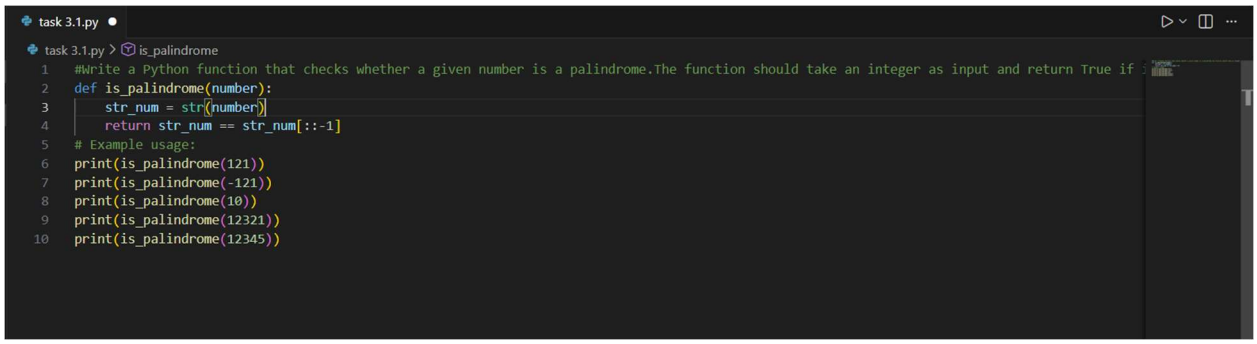
Assessment: 3.1

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

Prompt:

Write a Python function that checks whether a given number is a palindrome.The function should take an integer as input and return True if it is a palindrome,otherwise return False.

Code:



Output:

Siginificance:

The zero-shot prompt is significant because it tests the AI's basic understanding of the palindrome problem without giving any examples. It shows that the AI can generate a logically correct solution using only a task description. However, it also reveals that without guidance, the AI may miss important edge cases such as negative or invalid inputs. This demonstrates that zero-shot prompting is useful for simple problems but may not produce fully reliable code.

Task 2: One-Shot Prompting (Factorial Calculation)

Prompt:

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

Example:

Input: 5

Output: 120

The function should take an integer as input and return its factorial.

Code:

```python
#Write a Python function to compute the factorial of a given number.
#Example:
#Input: 5
#Output: 120
#The function should take an integer as input and return its factorial.
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers"
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Example usage:
print(factorial(5))  # Output: 120
print(factorial(0))  # Output: 1
print(factorial(-3)) # Output: Factorial is not defined for negative numbers
```

Output:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 3.1.py"
120
1
Factorial is not defined for negative numbers
PS C:\Users\shryi\Downloads\AI>
```

Zero-shot:

Code:

```python
#Write a Python function to calculate the factorial of a number.
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
# Example usage:
print(factorial(5))  # Output: 120
print(factorial(0))  # Output: 1
print(factorial(7))  # Output: 5040
print(factorial(-3))
```

Output:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 3.1.py"
120
1
5040
1
PS C:\Users\shryi\Downloads\AI>
```

Comparsion Table:

```
#Write a comparison table on both zero shot and one shot based on the above two implementations of factorial
| Feature                    | Zero-Shot Implementation                | One-Shot Implementation                          |
| -------------------------- | --------------------------------------- | ------------------------------------------------ |
| Prompt type                | Only instruction, no example            | One input□output example (5 → 120) provided      |
| Guidance level             | Low □ AI relies only on general knowledge | Higher □ example guides expected behavior      |
| Code structure             | Basic factorial loop                    | More structured and clear                        |
| Handling of positive numbers | Correctly computes factorial          | Correctly computes factorial                     |
| Handling of zero           | Returns 1                               | Returns 1                                        |
| Handling of negative numbers | Not handled in basic version          | Explicitly handled with message                  |
| Error handling             | Minimal or none                         | Better input validation                          |
| Readability                | Simple and short                        | Clearer and more user-friendly                   |
| Correctness                | Correct for standard positive inputs    | Correct for positive, zero, and negative         |
| Robustness                 | Moderate                                | Higher due to validation                         |
| Overall quality            | Good basic solution                     | More reliable and complete solution              |
```

Significance:

One-shot prompting is significant because the single example (5 → 120) clearly guides the AI about the expected behavior of the factorial function. It improves the clarity of logic and reduces misunderstanding compared to zero-shot prompting. The generated code is usually more accurate and better structured. This experiment shows that even one example can greatly improve the quality and reliability of AI-generated code.

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

Prompt:

Write a python program to check whether a given number is armstrong number or not

Ex:input:153 #Armstrong number

Ex:input:370 #Armstrong Number

Ex:input:123 #Not a armstrong number

Code:

```python
#Write a python program to check whether a given number is armstrong number or not
#Ex:input:153 #Armstrong number
#Ex:input:370 #Armstrong Number
#Ex:input:123 #Not a armstrong number
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number
# Example usage:
print(is_armstrong(153))  # Output: True
print(is_armstrong(370))  # Output: True
print(is_armstrong(123))  # Output: False
```

Output:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 3.1.py"
True
True
False
```

Significance:

Few-shot prompting is significant because providing multiple examples strongly guides the AI to understand both the logic and expected output format. The Armstrong and non-Armstrong examples help the AI correctly infer digit processing and power calculation. This results in more structured and accurate code compared to zero-shot and one-shot prompting. Testing with boundary and invalid inputs shows improved generalization and fewer logical errors, proving that multiple examples lead to more reliable solutions.

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

Prompt:

Write a optimized python program that classifies a given number as prime ,composite or neither

Code:

```python
#Write a optimized python program that classifies a given number as prime ,composite or neither
def classify_number(n):
    if n <= 1:
        return "Neither prime nor composite"
    elif n == 2:
        return "Prime"
    elif n % 2 == 0:
        return "Composite"
    else:
        for i in range(3, int(n**0.5) + 1, 2):
            if n % i == 0:
                return "Composite"
        return "Prime"
# Example usage:
print(classify_number(1))    # Output: Neither prime nor composite
print(classify_number(2))    # Output: Prime
print(classify_number(15))   # Output: Composite
```

Output:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 3.1.py"
Neither prime nor composite
Prime
Composite
PS C:\Users\shryi\Downloads\AI>
```

Siginificance:

Context-managed prompting is significant because clear instructions and constraints guide the AI to generate efficient and complete programs. By specifying input validation and optimization rules, the AI correctly handles edge cases like 0, 1, and negative numbers. It also produces optimized logic such as checking divisibility up to $\sqrt{n}$. Compared to earlier prompting strategies, this approach results in more professional and reliable code, proving that context is essential for complex tasks.

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

Prompt:

Write a python program to check whether a given number is perfect number or not

Code:

```
#Write a python program to check whether a given number is perfect number or not
def is_perfect_number(n):
    if n <= 1:
        return False
    divisors = [i for i in range(1, n) if n % i == 0]
    return sum(divisors) == n
# Example usage:
print(is_perfect_number(6))   # Output: True
print(is_perfect_number(28))  # Output: True
print(is_perfect_number(12))  # Output: False
```

Output:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 3.1.py"
True
True
False
```

Significance:

This task is significant because it evaluates the AI's ability to generate a perfect number checking program using only a task description. The AI usually produces correct basic logic, but inefficiencies such as unnecessary iterations or missing validation for non-positive inputs may appear. This shows that zero-shot prompting can generate working solutions but not optimized ones. It reinforces that zero-shot prompting is best suited for basic demonstrations rather than real-world applications.

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

Prompt:

Write a python program to determine whether a given number is even or odd

Examples:

Input: 8 → Output: Even

Input: 15 → Output: Odd

Input: 0 → Output: Even

Code:

```python
#Write a python program to determine whether a given number is even or odd
#Examples:
#• Input: 8 → Output: Even
#• Input: 15 → Output: Odd
#• Input: 0 → Output: Even
def is_even_or_odd(n):
    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Example usage:
print(is_even_or_odd(8))    # Output: Even
print(is_even_or_odd(15))   # Output: Odd
print(is_even_or_odd(0))    # Output: Even
```

Output:

```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shryi/Downloads/AI/task 3.1.py"
Even
Odd
Even
PS C:\Users\shryi\Downloads\AI>
```

Significance:

This task is significant because providing multiple examples, including edge cases like zero, helps the AI clearly understand expected outputs and validation needs. The generated program usually includes proper conditional checks and clear output messages. It also shows improved handling of negative numbers compared to zero-shot prompting. Testing with non-integer inputs demonstrates better robustness, proving that few-shot prompting greatly improves input handling and reliability.