

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

```
question 1
#write a python function that checks whether a given number is palindrome
#the function should return True if the number is palindrome otherwise
False
def is_palindrome(number):
    # Convert the number to string
    str_num = str(number)
    # Check if the string is equal to its reverse

    return str_num == str_num[::-1]
# Example usage
print(is_palindrome(121))  # True
print(is_palindrome(-121)) # False
```

Output: True

False

My analysis:

Within the zero-shot scenario, no examples are given to the AI. The resulting code is successful in reversing a number and comparing it to the original number, and it is correct for positive integers only. The solution is not addressing either the negative number or a non-integer input, and it underlines that zero-shot editing is successful in writing good core code, even if it may have problems with corner cases and editing inputs.

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

```
#write a python function to calculate the factorial of a given number
using recursion
def factorial(n):
    # Base case
    if n == 0 or n == 1:
        return 1
    # Recursive case
    return n * factorial(n - 1)
# Example usage
print(factorial(5))    # 120
print(factorial(0))    # 1
```

Output:

120

1

My analysis:

One-shot prompting allows both input and output examples, which aid in better comprehension of the task requirements by the AI model. The code chunks in one-shot prompting are more accurate than in zero-shot prompting and also contain a verification process for negative values. The provision of an example enhances both accuracy and clarity of the solution obtained.

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.

```
#write a python function to check whether a number is an armstrong number or not
def is_armstrong(number):
    # Convert the number to string to easily iterate over digits
    str_num = str(number)
    num_digits = len(str_num)
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    return sum_of_powers == number
# Example usage
print(is_armstrong(153)) # True
print(is_armstrong(123)) # False
```

Output:

True

False

My analysis:

Few shot prompts provide multiple input and output examples that enable the AI system to understand even the Armstrong number concept more clearly. This results in more precise code being developed, and it is able to cope with various situations properly because the understanding among the AI systems is enhanced by additional examples.

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.

```
#write an optimized python program to classify a number as:
# prime, composite or neither
def classify_number(n):
    if n <= 1:
        return "neither"
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
```

```
        return "composite"
    return "prime"
# Example usage
print(classify_number(11)) # prime
print(classify_number(15)) # composite
print(classify_number(1)) # neither
```

Output:

prime
composite
neither

My analysis:

Few shot prompting, or multiple examples, helped the AI understand the criteria by which it was supposed to classify prime, composite, or neither numbers. This resulted in well-written code that is efficient and able to correctly identify numbers based on examples given.

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.

```
#write a python function that check whether a given number is a perfect
number
def is_perfect_number(n):
    if n <= 1:
        return False
    sum_of_divisors = sum(i for i in range(1, n) if n % i == 0)
    return sum_of_divisors == n
# Example usage
print(is_perfect_number(6)) # True
print(is_perfect_number(28)) # True
print(is_perfect_number(12)) # False
```

Output:

True

True

False

My analysis:

Few-shot prompting with diverse examples enabled the AI to better understand perfect numbers, improving logic accuracy, edge-case handling, and overall reliability of the generated solution.

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.

```
#write a python program to determine whether a number is even or odd with
input validation
#examples:
#input: 4->output: Even
#input: 7->output: Odd
#input: 0->output: Even
def even_or_odd():
    while True:
        user_input = input("Enter a number: ")
        try:
            number = int(user_input)
            if number % 2 == 0:
                print("Even")
```

```
        else:
            print("Odd")
            break
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
# Example usage
even_or_odd()
# Example usage
even_or_odd()
# Example usage
even_or_odd()
# Example usage
even_or_odd()
# Example usage
even_or_odd()
```

Output:

```
Enter a number: 5
Odd
Enter a number: 8
Even
Enter a number: 15
Odd
Enter a number: -1
Odd
Enter a number: -6
Even
```

My analysis:

Few-Shot Prompting Mostly enhances the handling of inputs and the clarity of the output. The AI identifies even and odd numbers correctly with the inclusion of validation for invalid inputs. The examples given to the AI make it address boundary cases such as zero and negative numbers, hence making the program robust and user-friendly.