

AI ASSISTED CODING

ASSIGNMENT-3.5

Name: Sudireddy Asritha Chowdary

Hallticket No: 2303A52297

Batch: 45

Question 1: Zero-Shot Prompting (Leap Year Check)

Write a zero-shot prompt to generate a Python function that checks whether a given year is a leap year.

Week2 -

Task:

- Record the AI-generated code.
- Test with years like 1900, 2000, 2024.
- Identify logical flaws or missing conditions.

Prompt:

```
# write a function to check leap year
```

Code:

```
C:\> Users > matta > OneDrive > Desktop > AI Assisted Coding > Untitled-1.py > is_leap_year.py
1 def is_leap_year(year: int) -> bool:
2     if year % 400 == 0:
3         return True
4     if year % 100 == 0:
5         return False
6     return year % 4 == 0
7 print(is_leap_year(1900))
8 print(is_leap_year(2000))
9 print(is_leap_year(2024))
```

Output:

```
top/AI Assisted Coding/Untitled-1.py"
False
True
True
```

Justification:

This is a zero-shot prompt because no examples were provided. The generated logic is correct and handles all leap year rules. No logical flaws found.

Question 2: One-Shot Prompting (GCD of Two Numbers)

Write a one-shot prompt with one example to generate a Python function that finds the Greatest Common Divisor (GCD) of two numbers.

Example:

Input: 12, 18 → Output: 6

Task:

- Compare with a zero-shot solution.
- Analyze algorithm efficiency.

Prompt:

One-Shot:

if the input is 12,18 output will be 6 write a function

Code:

```
11  def gcd(a: int, b: int) -> int:
12      while b != 0:
13          a, b = b, a % b
14      return abs(a)
15  print(gcd(12,18))
16  print(gcd(100,75))
```

Output:

```
top/AI Assisted Coding/Untitled-1.py"
6
25
```

Zero Shot Prompt:

#write a function for gcd of two numbers

Code:

```
59
60     def gcd(a: int, b: int) -> int:
61         a, b = abs(a), abs(b)
62         while b != 0:
63             a, b = b, a % b
64         return a
65     print(gcd(12, 18))
66     print(gcd(5, 100))
67
```

Output:

```
top/AI Assisted Coding/Untitled-1.py"
6
5
```

Justification:

Providing one example improves clarity and efficiency. The algorithm runs in **O(log n)** time, which is optimal.

Question 3: Few-Shot Prompting (LCM Calculation)

Write a few-shot prompt with multiple examples to generate a Python function that computes the Least Common Multiple (LCM).

Examples:

- Input: 4, 6 → Output: 12
- Input: 5, 10 → Output: 10
- Input: 7, 3 → Output: 21

Task:

- Examine how examples guide formula selection.
- Test edge cases.

Prompt:

if the input is 4,6 output is 12 , input is 5,10 output is 10, input is 7,3 output is 21 write a function

Code:

```

17
18     def lcm(a: int, b: int) -> int:
19         if a == 0 or b == 0:
20             return 0 # LCM with 0 is usually taken as 0
21         # Using relationship: lcm(a, b) * gcd(a, b) = |a * b|
22         def gcd(x: int, y: int) -> int:
23             while y:
24                 x, y = y, x % y
25             return x
26
27         return abs(a * b) // gcd(a, b)
28     print(lcm(4,6))
29     print(lcm(5,10))
30     print(lcm[7,21])
31

```

Output:

```

c:\Program Files\Python37\python37.exe -c "import os; os.chdir('C:/Users/3/marcelo/OneDrive/Desktop')"; top/AI Assisted Coding/Untitled-1.py"
12
10
21
Process finished with exit code 0

```

Justification:

Multiple examples guide the model to use the $\text{LCM} = (\text{axb})/\text{GCD}$ formula Few-shot prompting improves correctness and formula selection.

Question 4: Zero-Shot Prompting (Binary to Decimal Conversion)

Write a zero-shot prompt to generate a Python function that converts a binary number to decimal.

Task:

- Test with valid and invalid binary inputs.
- Identify missing validation logic.

Prompt:

```
# write a function to convert binary number to decimal number
```

Code:

```

31
32     def binary_to_decimal(binary_str: str) -> int:
33         decimal = 0
34         for digit in binary_str:
35             decimal = decimal * 2 + int(digit)
36         return decimal
37     print(binary_to_decimal("1010"))
38

```

Output:

```
c:\Programs\Python\Python310\python.exe c:/Users/Smruti/OneDrive/Desktop/  
top/AI Assisted Coding/Untitled-1.py"  
10
```

Justification:

Zero-shot output is concise but lacks **input validation**, which reduces robustness.

Question 5: One-Shot Prompting (Decimal to Binary Conversion)

Write a one-shot prompt with an example to generate a Python function that converts a decimal number to binary.

Example:

Input: 10 → Output: 1010

Task:

- Compare clarity with zero-shot output.
- Analyze handling of zero and negative numbers.

Prompt:

input is 10 and output is 1010 write a function

Code:

```
38  
39  def decimal_to_binary(n: int) -> str:  
40      if n == 0:  
41          return "0"  
42      bits = ""  
43      while n > 0:  
44          bits = str(n % 2) + bits  
45          n //= 2  
46      return bits  
47 print(decimal_to_binary(10))  
48
```

Output:

```
top/AI Assisted Coding/Untitled-1.py"  
1010
```

Justification:

One example improves clarity compared to zero-shot. Negative numbers are not handled explicitly, which is a limitation.

Question 6: Few-Shot Prompting (Harshad Number Check)

Write a few-shot prompt to generate a Python function that checks whether a number is a Harshad (Niven) number.

Examples:

- Input: 18 → Output: Harshad Number
- Input: 21 → Output: Harshad Number
- Input: 19 → Output: Not a Harshad Number

Task:

- Test boundary conditions.
- Evaluate robustness

Prompt:

```
# input is 18 then output Harshad Number,input is 21 output Harshad Number  
write a code
```

Code:

```
19  def check_harshad_number(n: int):  
20      digit_sum = sum(int(digit) for digit in str(n))  
21      if digit_sum != 0 and n % digit_sum == 0:  
22          print("Harshad Number")  
23      else:  
24          print("Not a Harshad Number")  
25  check_harshad_number(18)  
26  check_harshad_number(21)  
27  check_harshad_number(19)  
28
```

Output:

```
top/AI Assisted Coding/Untitled-1.py"  
Harshad Number  
Harshad Number  
Not a Harshad Number
```

Justification:

Few-shot examples clearly define the rule. Edge case for 0 is missing, slightly reducing robustness.