

AI ASSISTED CODING

ASSIGNMENT – 3.5

M. Vennela

Roll No : (2303A51112)

BATCH-03

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.

The screenshot shows a code editor window with the following Python code:

```
# generate a python function that checks if a year is a leap year or not
def is_leap_year(year):
    """Check if a given year is a leap year.

    A year is a leap year if it is divisible by 4,
    except for end-of-century years, which must be divisible by 400.

    Args:
        year (int): The year to check.
    Returns:
        bool: True if the year is a leap year, False otherwise.
    """
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

# Example usage:
year = 1900
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")

# Running the code
if __name__ == "__main__":
    print(is_leap_year(2024))
    print(is_leap_year(1900))
    print(is_leap_year(2000))

# Output
2024 is a leap year.
1900 is not a leap year.
2000 is a leap year.

[User] exited with status in 0.48 seconds
```

The code defines a function `is_leap_year` that takes a year as input and returns `True` if it is a leap year, and `False` otherwise. It handles the rules for leap years: divisible by 4, except for end-of-century years which must be divisible by 400. The code then tests this function with years 2024, 1900, and 2000, printing the results to the console.

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.

The screenshot shows a code editor with a dark theme. In the top-left pane, there is a file named 'Untitled-4' containing the following Python code:

```
num = 12, 18
gcd = 0

def compute_gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

if __name__ == "__main__":
    a = 12
    b = 18
    print("Input -> ", (a, b))
    print("GCD -> ", compute_gcd(a, b))
```

In the bottom-left pane, the terminal window shows the output of running the script:

```
[Running] python3 -u "C:\Users\Yash\Downloads\Untitled-4.py"
num = (12, 18)
gcd = 6

[Done] exited with status 0 in 0.427 seconds
```

A sidebar on the right is titled 'Ask about your code' with the sub-instruction 'All responses may be incomplete. Generate Agent instructions to comment AI with your confidence.'

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.

```
#!/usr/bin/python
# This program finds the greatest common divisor of two numbers.
# It uses Euclid's algorithm which repeatedly divides the larger number by the smaller one
# until the remainder is zero. The last non-zero remainder is the gcd.
# Example: gcd(48, 18) = 6

def gcd(a, b):
    if a < b:
        greater = a
        less = b
    else:
        greater = b
        less = a

    while True:
        if greater % less == 0:
            gcd = less
            break
        greater -= less
        greater = greater % less
    return gcd

print(gcd(48, 18))
```

The screenshot shows a Jupyter Notebook interface with several tabs at the top. The active tab contains the provided Python code. To the right of the code, there is a "Ask about your code" button with a speech bubble icon. Below the code, the terminal output shows the execution of the script and its result. At the bottom, there are buttons for "Run", "Kernel", "Help", and "File".

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.

```
# generate a python function that converts binary number to decimal
def binary_to_decimal(binary_str):
    decimal_value = 0
    binary_str = binary_str[1:-1] # remove the quotes to process from input significant bits
    for index, digit in enumerate(binary_str):
        if digit == '1':
            decimal_value += 2 ** index
    return decimal_value

# Example usage:
binary_number = "1001"
decimal_number = binary_to_decimal(binary_number)
print("The decimal value of binary (%s) is (%d)" % (binary_number, decimal_number))

# Running the code in a terminal
[Running] Python in "C:\Users\valentyn\PycharmProjects\CopyCalculator\src\python"
the decimal value of binary 1001 is 9
[Done] exited with status 0 in 4.063 seconds
```

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.

A screenshot of a code editor window. The top bar shows tabs for 'File', 'Edit', 'Search', 'View', 'Ctrl', 'Run', 'Server', 'Terminal', and 'Help'. The main area contains the following Python code:

```
num = 10
binary_number = bin(num)

def decimal_to_binary(n):
    if n > 1:
        decimal_to_binary(n // 2)
        print(n % 2, end=' ')
    else:
        decimal_to_binary(n)

decimal_to_binary(10)
```

The bottom terminal window shows the command 'python -i 10' being run, and the output '1010' is displayed. A tooltip 'Ask about your code' is visible on the right side of the interface.

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**

A screenshot of a code editor window showing a Python script named `is_narcissistic.py`. The script defines a function `is_narcissistic` that takes a number `n` and returns `True` if it is a narcissistic number (the sum of its digits raised to the power of the number of digits equals the number itself) and `False` otherwise. It also includes a check function `check` that tests the function with a set of predefined numbers.

```
1 #!/usr/bin/python
2
3 def is_narcissistic(n):
4     sum = 0
5     for digit in str(n):
6         sum += int(digit)**len(str(n))
7     return sum == n
8
9 def check():
10    test_numbers = [19, 37, 120]
11    for num in test_numbers:
12        if is_narcissistic(num):
13            print("%d is narcissistic" % num)
14        else:
15            print("%d is not narcissistic" % num)
16
17 check()
```

The terminal output shows the script running and printing the results for the test numbers:

```
[running] python -m is_narcissistic
19 is narcissistic
37 is narcissistic
120 is not narcissistic
```