SR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Course Title – AI Assisted Coding

Course Code : 24CS002PC215

Assignment Type : Lab

Academic Year : 2025–2026

Name : Soumith Sripathi

Enroll Number : 2403A51340

Batch Number : 24BTCAICSB14

Week – 1

Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI

Task Description –1

Question : Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

Expected Output – 1
Functional code with correct output and screenshot.

CODE :

```
def calculate_stats(numbers):
    mean = sum(numbers) / len(numbers)
    minimum = min(numbers)
    maximum = max(numbers)
    return mean, minimum, maximum

# Example usage:
numbers = [10, 20, 30, 40, 50]
mean, minimum, maximum = calculate_stats(numbers)
print(f"Mean: {mean}, Min: {minimum}, Max: {maximum}")
```

OUTPUT :

```
List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50
The list is empty.
```

Explanation of the above Code :

~ A function named analyze_numbers is defined.

~ It takes one input parameter: numbers, which is expected to be a list of numeric values (integers or floats)

~ This checks if the numbers list is empty.

~ If it's empty, the function returns None for all three values, because mean, min, and max are undefined for an empty list.

~ The mean is calculated by dividing the sum of all elements by the number of elements.

~ min(numbers) finds the smallest number in the list.

~ max(numbers) finds the largest number.

~ The function returns all three values as a tuple: (mean, min, max)

~ A list of numbers is defined.

Task Description – 2

Question : Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

CODE :

```
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    result = sum(int(digit)**num_digits for digit in num_str)
    return result == number

def is_armstrong_number(n):
    digits = [int(d) for d in str(n)]
    power = len(digits)
    return n == sum([d ** power for d in digits])

print(is_armstrong(153))          # Gemini version
print(is_armstrong_number(153))   # Copilot version
print(is_armstrong(123))          # Gemini version
print(is_armstrong_number(123))   # Copilot version
```

OUTPUT :

```
Output

True
True
False
False
```

Expected Output :

Side–by–side comparison table with observations and screenshots.

Aspect	Gemini Output	Copilot Output
Function Name	`is_armstrong`	`is_armstrong_number`
Input Parameter	`number`	`n`
Steps Used	- Converts number to string - Gets digit count - Sums digits raised to the power of count - Compares sum to original	- Converts number to string - Gets digit count - Sums digits raised to the power of count - Compares sum to original
Return Value	Boolean (True/False)	Boolean (True/False)
Code Style	Uses generator expression in `sum`	Uses list comprehension in `sum`
Variable Naming	`num_str`, `num_digits`, `result`	`digits`, `power`
Output Example	`is_armstrong(153) → True`	`is_armstrong_number(153) → True`

Explanation of the above Code :

~ An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits.

For 153 is a armstrong :

1. num_str = "153" (converted to string)

2. num_digits = 3

3. result = $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

4. result == number → 153 == 153 → True

For 123 is not a armstrong :

$1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36$

$36 \neq 123$ → False

Task Description –3

Question : Ask Gemini to explain a Python function (e.g., is_prime(n) or is_palindrome(s)) line by line. Choose either a prime–checking or palindrome–checking function and document the explanation provided by Gemini.

CODE :

```
def is_palindrome(s):
    s = s.lower()  # Convert the string to lowercase for case-insensitive com
    s = ''.join(c for c in s if c.isalnum())  # Remove non-alphanumeric chara
    return s == s[::-1]  # Check if the string is equal to its reverse

# Example usage:
print(is_palindrome("Madam"))         # Output: True
print(is_palindrome("Hello"))         # Output: False
print(is_palindrome("A man, a plan, a canal: Panama"))  # Output: True
```

OUTPUT :

```
True
False
True
```

Explanation of the above Code :

~ A function named is_palindrome is defined.

~ It takes a single parameter s, which is the input string.

~ This ensures the comparison is not case–sensitive.

~ This removes all characters that are not letters or digits.

~ s[::–1] is Python slicing to reverse the string.

~ The function returns True if the original and reversed strings match.

For madam is a palindrome :

"madam" → "madam" == madam[::–1] → True

For hello is not a palindrome :

"hello" → "hello" == olleh" → False

Task Description – 4 :

Question : Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output.

CODE :

```
def sum_of_natural_numbers(n):
  if n < 0:
    return "Input must be a non-negative integer."
  else:
    return n * (n + 1)
N = 10
sum_n = sum_of_natural_numbers(N)
print(f"The sum of the first {N} natural numbers is: {sum_n}")
N = 0
sum_n = sum_of_natural_numbers(N)
print(f"The sum of the first {N} natural numbers is: {sum_n}")
N = 5
sum_n = sum_of_natural_numbers(N)
print(f"The sum of the first {N} natural numbers is: {sum_n}")
N = -5
sum_n = sum_of_natural_numbers(N)
print(f"The sum of the first {N} natural numbers is: {sum_n}")
```

OUTPUT :

```
The sum of the first 10 natural numbers is: 55
The sum of the first 0 natural numbers is: 0
The sum of the first 5 natural numbers is: 15
The sum of the first -5 natural numbers is: Input must be a non-negative integer.
```

Explanation of the above Code :

~ A function named sum_of_natural_numbers is defined.

~ It takes one parameter n, which represents how many natural numbers to sum
(e.g., n = 5 → sum of 1 + 2 + 3 + 4 + 5).

~ If n is negative (e.g., −5), the function returns an error message.

~ Natural numbers start from 1, so negative values are invalid.

Case 1:
N = 10
Formula used: 10 * 11 = 110
(Incorrect due to missing /2)
Correct sum: 55, but code will return 110

Case 2: N = 0
N = 0
0 * 1 = 0 → Output: 0
(Correct)

Task Description – 5:

Question : write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.

CODE :

```
def sum_odd_even(numbers):
    odd_sum = 0
    even_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return odd_sum, even_sum

# Example usage:
numbers = (1, 1, 3)
odd_sum, even_sum = sum_odd_even(numbers)
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

OUTPUT :

Output:

```
Sum of odd numbers: 5
Sum of even numbers: 0
```

Explanation of the above Code :

~ This line defines a function called sum_odd_even that takes one parameter numbers.

~ numbers is expected to be a list or tuple of integers.

~ odd_sum to keep track of the total of odd numbers.

~ even_sum to keep track of the total of even numbers.

~ A loop starts to go through each number in the input numbers.

~ the number is even (remainder is 0 when divided by 2).

~ For numbers = (1, 1, 3):

All numbers are odd: 1 + 1 + 3 = 5
No even numbers → sum is 0