Assignment = 02

Name: Pothunuri Likhitha

Roll no: 2403A510D1

Batch no: 05

Subject: AI assisted coding

Task Description 1

● 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

● Functional code with correct output and screenshot

Code:

```
def calculate_stats(numbers):
  """
  Calculates the mean, minimum, and maximum of a list of numbers.

  Args:
    numbers: A list of numbers.

  Returns:
    A tuple containing the mean, minimum, and maximum values.
  """
  if not numbers:
    return None, None, None  # Handle empty list case

  mean = sum(numbers) / len(numbers)
  minimum = min(numbers)
  maximum = max(numbers)
```

```python
    minimum = min(numbers)
    maximum = max(numbers)

    return mean, minimum, maximum

# Example usage:
my_list = [1, 5, 2, 8, 3, 9, 4, 6, 7]
mean_value, min_value, max_value = calculate_stats(my_list)

print(f"List: {my_list}")
print(f"Mean: {mean_value}")
print(f"Minimum: {min_value}")
print(f"Maximum: {max_value}")
```

Output:

List: [1, 5, 2, 8, 3, 9, 4, 6, 7]
Mean: 5.0
Minimum: 1
Maximum: 9

Task Description 2

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

Expected Output 2

● Side-by-side comparison table with observations and screenshots

Gemini Code:

```python
def is_armstrong(number):
  """
  Checks if a number is an Armstrong number.

  Args:
    number: An integer.

  Returns:
    True if the number is an Armstrong number, False otherwise.
  """
  # Convert the number to a string to get the number of digits
  num_str = str(number)
  num_digits = len(num_str)
  sum_of_powers = 0
  # Calculate the sum of the digits raised to the power of the number of digits
  for digit in num_str:
    sum_of_powers += int(digit) ** num_digits

  # Check if the sum is equal to the original number
  return sum_of_powers == number

# Example usage:
```

```python
# Example usage:
num1 = 153
num2 = 123

print(f"Is {num1} an Armstrong number? {is_armstrong(num1)}")
print(f"Is {num2} an Armstrong number? {is_armstrong(num2)}")
```

Copilot Code:

```python
def is_armstrong(n: int) -> bool:
    """
    Return True if n is an Armstrong (narcissistic) number in base-10.
    Assumes n is a non-negative integer.
    """
    s = str(n)
    k = len(s)
    total = sum(int(ch) ** k for ch in s)
    return total == n

def main():
    raw = input("Enter a non-negative integer: ").strip()
    if not raw.isdigit():
        print("Invalid input. Please enter digits only.")
        return

    n = int(raw)
    if n < 0:
        print("Please enter a non-negative integer.")
        return
```

```
    n = int(raw)
    if n < 0:
        print("Please enter a non-negative integer.")
        return

    if is_armstrong(n):
        print(f"{n} is an Armstrong number.")
    else:
        print(f"{n} is not an Armstrong number.")

if __name__ == "__main__":
    main()
```

Gemini Output:

Is 153 an Armstrong number? True

Is 123 an Armstrong number? False

Copilot output:

Input	Digits k	Sum of powers	Output
0	1	$0^1 = 0$	Armstrong
5	1	$5^1 = 5$	Armstrong
10	2	$1^2 + 0^2 = 1$	Not Armstrong
153	3	$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$	Armstrong
370	3	$3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370$	Armstrong
371	3	$3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$	Armstrong
407	3	$4^3 + 0^3 + 7^3 = 64 + 0 + 343 = 407$	Armstrong
9474	4	$9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474$	Armstrong
123	3	$1^3 + 2^3 + 3^3 = 36$	Not Armstrong

As a part of the task we have to compare the differences in both(Gemini code and Microsoft copilot code) Codes. Let's breakout the task.

1. Function Signature & Documentation

Aspect	Gemini Code	Microsoft Copilot Code
Function name & parameters	def is_armstrong(number) — no type hints.	def is_armstrong(n: int) -> bool — has type hints for input and return type.
Docstring style	Explains purpose, arguments, and return type in a simple comment block.	Explains purpose with clear assumptions (n is non-negative) in concise form.

Difference:

Copilot code uses **Python type hints** and a more concise docstring; Gemini's is more verbose but beginner-friendly.

2. Core Logic Implementation

Step	**Gemini Code**	**Microsoft Copilot Code**
Convert to string	num_str = str(number)	s = str(n)
Find number of digits	num_digits = len(num_str)	k = len(s)
Sum of digits to the power of digit count	Uses a for loop:	

Task Description 3

● 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.
Expected Output

● Detailed explanation with the code snippet and Gemini's response

Code:

```python
def is_prime(n):
  """
  Checks if a number is prime.

  Args:
    n: An integer.

  Returns:
    True if the number is prime, False otherwise.
  """
  if n <= 1:
    return False  # Numbers less than or equal to 1 are not prime
  for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
      return False  # If the number is divisible by any number from 2 to the square root, it's not prime
  return True  # If the number is not divisible by any of the above, it's prime
```

```python
  return True  # If the number is not divisible by any of the above, it's prime

# Example usage:
num = 17
print(f"{num} is prime: {is_prime(num)}")

num = 10
print(f"{num} is prime: {is_prime(num)}")
```

Output:

17 is prime: True
10 is prime: False

Explanation:

This Python function, is_ prime(n), checks if a given integer n is a prime number. Here's how it works:

- **if n <= 1: return False**: It first handles the base cases. Numbers less than or equal to 1 are not considered prime, so it immediately returns False.

- **`for I   in range (2, int (n**0.5) + 1): **: This line starts a loop. It iterates through numbers starting from 2 up to the square root of n (including the square root). We only need to check for divisors up to the square root because if a number` has a divisor greater than its square root, it must also have a divisor smaller than its square root.
- **if n % I == 0: return False**: Inside the loop, for each number I , it checks if n is divisible by I using the modulo operator (%). If the remainder of the division is 0, it means n is divisible by I , and therefore, n is not a prime number. In this case, the function immediately returns False.
- **return True**: If the loop finishes without finding any divisors (meaning the if n % i == 0: condition was never met), it means n is not divisible by any number from 2 up to its square root. This indicates that n is a prime number, and the function returns True.

In simple terms, the function tries to divide the input number by all possible integers starting from 2 up to its square root. If it finds any number that divides it evenly, it's not prime. If it goes through all these checks without finding a divisor, then the number is prime.

● Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output.
● Optionally, compare Cursor AI's generated code with Gemini's output.

Task Description 4

Expected Output 4
● Screenshots of Cursor AI setup, prompts used, and generated code with output.

Cursor AI Code:

```python
def sum_natural_numbers(n: int) -> int:
    """
    Returns the sum of the first n natural numbers.
    Uses the formula n*(n+1)//2 for efficiency.
    """
    if n < 1:
        return 0
    return n * (n + 1) // 2

# Example test
if __name__ == "__main__":
    N = 10
    print(f"Sum of first {N} natural numbers: {sum_natural_numbers(N)}")
```

Gemini AI Code:

Commands + Code + Text ▷ Run all

```python
def sum_of_natural_numbers(n):
  """
  Calculates the sum of the first N natural numbers.

  Args:
    n: A positive integer.

  Returns:
    The sum of the first N natural numbers.
  """
  if n < 1:
    return "Input must be a positive integer"
  return n * (n + 1) // 2

# Example usage:
num = 10
sum_n = sum_of_natural_numbers(num)
print(f"The sum of the first {num} natural numbers is: {sum_n}")

num = 5
sum_n = sum_of_natural_numbers(num)
print(f"The sum of the first {num} natural numbers is: {sum_n}")
```

Cursor AI Output:

Sum of first 10 natural numbers: 55

Gemini AI Output:

The sum of the first 10 natural numbers is: 55

The sum of the first 5 natural numbers is: 15

Comparision Table:

Feature	Cursor AI	Gemini
Approach	Formula-based (n*(n+1)//2)	Loop-based sum
Efficiency	O(1) time	O(n) time
Readability	Short, efficient	Easy for beginners
Math Concept	Requires formula knowledge	No formula needed

Task Description 5

● Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.

● Refactor the code to improve logic and readability.

Expected Output 5

● Student-written refactored code with explanations and output screenshots

Code:

```python
def sum_odd_even(numbers):
  """
  Calculates the sum of odd and even numbers in a tuple.

  Args:
    numbers: A tuple of numbers.

  Returns:
    A tuple containing the sum of odd numbers and the sum of even numbers.
  """
  sum_odd = 0
  sum_even = 0
  for number in numbers:
    if number % 2 == 0:
      sum_even += number
    else:
      sum_odd += number
  return sum_odd, sum_even

# Example usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
odd_sum, even_sum = sum_odd_even(my_tuple)

print(f"Tuple: {my_tuple}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

Output:

Tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of odd numbers: 25
Sum of even numbers: 30