LAB ASSIGNMENT-2

NAME: janagani Varshitha

ROLL – 2403a54085

Batch – 03 (DS)

TASK 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.

PROMPT: Write a function in python to display mean,minimum,maximum from the given list

CODE:

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

  Args:
    data: A list of numbers.
  """
  if not data:
    print("The list is empty.")
    return

  minimum = min(data)
  maximum = max(data)
  mean = sum(data) / len(data)

  print(f"Minimum: {minimum}")
  print(f"Maximum: {maximum}")
  print(f"Mean: {mean}")

# Example usage:
my_list = [10, 20, 30, 40, 50]
analyze_list(my_list)

empty_list = []
analyze_list(empty_list)
```

```
Minimum: 10
Maximum: 50
Mean: 30.0
The list is empty.
```

OUTPUT:

```
Minimum: 10
Maximum: 50
Mean: 30.0
The list is empty.
```

EXPLANATION:

1. Checks for an empty list: It first checks if the input list data is empty. If it is, it prints a message indicating the list is empty and stops.
2. Calculates statistics: If the list is not empty, it calculates the minimum value using min(data), the maximum value using max(data), and the mean (average) by summing all the elements with sum(data) and dividing by the number of elements with len(data).
3. Displays results: Finally, it prints the calculated minimum, maximum, and mean values using formatted strings (f-strings).
4. Example Usage: The code then shows two examples of how to use the analyze_list function: one with a list of numbers and one with an empty list.

TASK-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

PROMPT: write a function in python which checks whether a number is an Armstrong number

CODE:

```
# Convert the number to a string to get the number of digits
num_str = str(number)
num_digits = len(num_str)
armstrong_sum = 0

# Calculate the sum of each digit raised to the power of the number of digits
for digit in num_str:
  armstrong_sum += int(digit) ** num_digits

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

```
# Example usage:
num_to_check = 153
if is_armstrong_number(num_to_check):
  print(f"{num_to_check} is an Armstrong number.")
else:
  print(f"{num_to_check} is not an Armstrong number.")

num_to_check = 123
if is_armstrong_number(num_to_check):
  print(f"{num_to_check} is an Armstrong number.")
else:
  print(f"{num_to_check} is not an Armstrong number.")
```

```
153 is an Armstrong number.
123 is not an Armstrong number.
```

OUTPUT:

```
153 is an Armstrong number.
123 is not an Armstrong number.
```

EXPLANATION:

1. Function Definition: def is_armstrong_number(number): defines a function that takes one argument, number.
2. Docstring: The triple-quoted string explains what the function does, the definition of an Armstrong number, its arguments, and what it returns.
3. Convert to String: num_str = str(number) converts the input number to a string to easily access individual digits and determine the number of digits.
4. Number of Digits: num_digits = len(num_str) calculates the number of digits in the number.
5. Initialize Sum: armstrong_sum = 0 initializes a variable to store the sum of the digits raised to the power of the number of digits.
6. Iterate Through Digits: The for digit in num_str: loop iterates through each character (digit) in the number string.
7. Calculate Sum of Powers: armstrong_sum += int(digit) ** num_digits converts each digit back to an integer, raises it to the power of the total number of digits, and adds it to armstrong_sum.
8. Check for Armstrong Number: return armstrong_sum == number compares the calculated armstrong_sum with the original number. If they are equal, the function returns True, indicating it's an Armstrong number; otherwise, it returns False.

TASK-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.

PROMPT: write a function in python which checks the number is prime or not using is_prime() and also checks if the number is palindrome or not using is_palindrome() and display the output

CODE:

```
def is_prime(num):
    """Checks if a number is prime."""
    if num <= 1:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def is_palindrome(num):
    """Checks if a number is a palindrome."""
    return str(num) == str(num)[::-1]

def check_number(num):
    """Checks if a number is prime and/or a palindrome and displays the output."""
    prime_check = is_prime(num)
    palindrome_check = is_palindrome(num)

    print(f"The number {num} is:")
    if prime_check and palindrome_check:
        print(" - Prime and a Palindrome")
    elif prime_check:
        print(" - Prime")
    elif palindrome_check:
        print(" - a Palindrome")
    else:
        print(" - Neither Prime nor a Palindrome")
```

```
# Example usage:
number_to_check = 131
check_number(number_to_check)

number_to_check = 12
check_number(number_to_check)

number_to_check = 121
check_number(number_to_check)

number_to_check = 10
check_number(number_to_check)
```

```
The number 131 is:
 - Prime and a Palindrome
The number 12 is:
 - Neither Prime nor a Palindrome
The number 121 is:
 - a Palindrome
The number 10 is:
 - Neither Prime nor a Palindrome
```

OUTPUT:


```
The number 131 is:
 - Prime and a Palindrome
The number 12 is:
 - Neither Prime nor a Palindrome
The number 121 is:
 - a Palindrome
The number 10 is:
 - Neither Prime nor a Palindrome
```


EXPLANATION:

1. is_prime(num) function:
    - Takes an integer num as input.

- Checks if the number is less than or equal to 1. If so, it's not prime, and the function returns False.
- It then iterates through numbers starting from 2 up to the square root of num. We only need to check 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.
- In each iteration, it checks if num is divisible by the current number i. If it is, num is not prime, and the function returns False.
- If the loop completes without finding any divisors, the number is prime, and the function returns True.

2. is_palindrome(num) function:
    - Takes an integer num as input.
    - Converts the number to a string using str(num).
    - Compares the string representation of the number with its reverse (str(num)[::-1]). The slicing [::-1] creates a reversed copy of the string.
    - If the string and its reverse are equal, the number is a palindrome, and the function returns True; otherwise, it returns False.
3. check_number(num) function:
    - Takes an integer num as input.
    - Calls the is_prime() function to check if the number is prime and stores the result in prime_check.
    - Calls the is_palindrome() function to check if the number is a palindrome and stores the result in palindrome_check.
    - Prints a statement indicating which checks are true for the given number:
        - If both prime_check and palindrome_check are True, it prints that the number is "Prime and a Palindrome".
        - If only prime_check is True, it prints that the number is "Prime".
        - If only palindrome_check is True, it prints that the number is "a Palindrome".
        - If neither is True, it prints that the number is "Neither Prime nor a Palindrome".

TASK-4: 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

PROMPT: write a function python which takes input and prints the sum of first n natural numbers

CODE:

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

  Args:
    n: The number of natural numbers to sum (a non-negative integer).
  """
  if n < 0:
    print("Please enter a non-negative integer.")
    return

  # Using the formula for the sum of the first n natural numbers: n * (n + 1) / 2
  sum_n = n * (n + 1) // 2
  print(f"The sum of the first {n} natural numbers is: {sum_n}")

# Example usage:
input_number = 10
sum_of_natural_numbers(input_number)

input_number = 5
sum_of_natural_numbers(input_number)

input_number = -3
sum_of_natural_numbers(input_number)
```

```
The sum of the first 10 natural numbers is: 55
The sum of the first 5 natural numbers is: 15
Please enter a non-negative integer.
```

OUTPUT:

```
The sum of the first 10 natural numbers is: 55
The sum of the first 5 natural numbers is: 15
Please enter a non-negative integer.
```

EXPLANATION:

1. **Function Definition**: def sum_of_natural_numbers(n): defines a function that takes one argument, n, which represents the number of natural numbers to sum.
2. **Docstring**: The triple-quoted string explains what the function does, its arguments, and what it calculates.
3. **Input Validation**: if n < 0: checks if the input number n is negative. Natural numbers are positive integers, so if n is negative, the code prints an error message and uses return to exit the function without further execution.
4. **Calculate Sum**: sum_n = n * (n + 1) // 2 calculates the sum of the first n natural numbers using the mathematical formula: n * (n + 1) / 2. The // operator is used for integer division to ensure the result is an integer.
5. **Print Output**: print(f"The sum of the first {n} natural numbers is: {sum_n}") prints the calculated sum in a formatted string.

- **TASK-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.

PROMPT: write a python program to calculate sum of odd and even numbers separated and display it

CODE:

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

  Args:
    numbers: A list 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 with a list:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
odd_sum, even_sum = sum_odd_even(my_list)
print(f"Sum of odd numbers in the list: {odd_sum}")
print(f"Sum of even numbers in the list: {even_sum}")

# Example usage with a range:
my_range = range(1, 11) # Numbers from 1 to 10
odd_sum_range, even_sum_range = sum_odd_even(my_range)
print(f"\nSum of odd numbers in the range: {odd_sum_range}")
print(f"Sum of even numbers in the range: {even_sum_range}")
```

OUTPUT:

```
Sum of odd numbers in the list: 25
Sum of even numbers in the list: 30

Sum of odd numbers in the range: 25
Sum of even numbers in the range: 30
```

EXPLANATION:

1. Function Definition: def sum_odd_even(numbers): defines a function that takes one argument, numbers, which is expected to be an iterable (like a list or range) containing numbers.
2. Docstring: The triple-quoted string explains what the function does, its arguments, and what it returns.

3. Initialize Sums: sum_odd = 0 and sum_even = 0 initialize two variables to store the cumulative sum of odd and even numbers, respectively.
4. Iterate Through Numbers: The for number in numbers: loop iterates through each number in the input numbers collection.
5. Check for Even or Odd: if number % 2 == 0: checks if the current number is even by using the modulo operator (%). If the remainder when divided by 2 is 0, the number is even. Otherwise, it's odd.
6. Accumulate Sums:
    - If the number is even, sum_even += number adds the number to the sum_even.
    - If the number is odd, sum_odd += number adds the number to the sum_odd.
7. Return Sums: return sum_odd, sum_even returns a tuple containing the final calculated sum of odd numbers and the sum of even numbers.