

ASSIGNMENT – 2.2

Thumma Hasini | 2303A52076 | Batch – 37

Task 1 : Use Gemini in Colab to generate a function that filters out all negative numbers from a list.

Prompt Used : Write a python code to separate all the negative numbers from a user given list of numbers and show both the lists

The screenshot shows a Google Colab notebook titled "Untitled5.ipynb". The code cell contains a Python function named `separate_negative_numbers` which takes a list of numbers and returns two lists: one with negative numbers and one with non-negative numbers. The function uses list comprehensions to filter the input list based on the sign of each number. The code also includes several examples demonstrating how to use the function with different types of input lists (mixed, positive only, negative only, and empty).

```
def separate_negative_numbers(numbers_list):
    """
    Separates negative numbers from a list and returns two lists: one with negative numbers
    and one with non-negative numbers.

    Args:
        numbers_list (list): A list of numerical values.

    Returns:
        tuple: A tuple containing two lists (negative_numbers, non_negative_numbers).
    """
    negative_numbers = [num for num in numbers_list if num < 0]
    non_negative_numbers = [num for num in numbers_list if num >= 0]
    return negative_numbers, non_negative_numbers

# Example usage with a mixed list:
my_list = [10, -5, 2, -15, 0, -3, 7]
negative_nums, non_negative_nums = separate_negative_numbers(my_list)
print("Original list: ", my_list)
print("Negative numbers: ", negative_nums)
print("Non-negative numbers: ", non_negative_nums)

# Example usage with a list containing only non-negative numbers:
positive_only_list = [1, 2, 3, 4, 5]
negatives_from_positive, non_negatives_from_positive = separate_negative_numbers(positive_only_list)
print("\nOriginal list: ", positive_only_list)
print("Negative numbers: ", negatives_from_positive)
print("Non-negative numbers: ", non_negatives_from_positive)

# Example usage with a list containing only negative numbers:
negative_only_list = [-1, -2, -3]
negatives_from_negative, non_negatives_from_negative = separate_negative_numbers(negative_only_list)
print("\nOriginal list: ", negative_only_list)
print("Negative numbers: ", negatives_from_negative)
print("Non-negative numbers: ", non_negatives_from_negative)

# Example usage with an empty list:
empty_list = []
negatives_from_empty, non_negatives_from_empty = separate_negative_numbers(empty_list)
print("\nOriginal list: ", empty_list)
print("The list is empty.")
print("Negative numbers: ", negatives_from_empty)
print("Non-negative numbers: ", non_negatives_from_empty)
```

The output cell shows the results of running the code with various test cases. The Gemini AI interface is visible on the right, with a prompt asking for a Python code to separate negative numbers and a message indicating it has updated the existing code to provide more comprehensive examples.

Explanation :

In this task, I removed unwanted negative values from sensor readings to make the data clean and usable. Negative readings can occur due to sensor errors or noise, so filtering them

improves accuracy. I used a simple function to check each value and keep only valid readings. Finally, I displayed the list before and after cleaning to verify the result clearly.

Task 2 : Use Gemini to generate a Python function that counts vowels, consonants, and digits in a string.

Prompt Used : Create a Python program using a function to count vowels, consonants, digits, and special characters from a given string.

Ignore case when checking vowels.

Display output neatly using print statements like:

Vowels =

Consonants =

Digits =

Special Characters =

Use input string: "AI Lab 2026!!"

The screenshot shows a Jupyter Notebook cell with the following code:

```
def count_all_chars(input_string):
    """
    Counts vowels, consonants, digits, and special characters in a string.
    Ignores case for vowels.

    Args:
        input_string (str): The string to analyze.

    Returns:
        tuple: A tuple containing (vowel_count, consonant_count, digit_count, special_char_count).
    """
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0
    special_char_count = 0

    for char in input_string:
        char_lower = char.lower()
        if char_lower.isalpha():
            if char_lower in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char_lower.isdigit():
            digit_count += 1
        else:
            # Consider anything that is not alpha or digit as a special character
            # You might want to refine this to exclude whitespace if needed,
            # but for 'AI Lab 2026!!' whitespace is not a special char
            if not char.isspace():
                special_char_count += 1

    return vowel_count, consonant_count, digit_count, special_char_count

# Input string:
input_str = "AI Lab 2026!!"

# Call the function
v, c, d, s = count_all_chars(input_str)

# Display output neatly
print("Input String: ", input_str)
print("Vowels = ", v)
print("Consonants = ", c)
print("Digits = ", d)
print("Special Characters = ", s)
```

Below the code, the output is shown:

```
Input String: AI Lab 2026!!
Vowels = 3
Consonants = 2
Digits = 4
Special Characters = 2
```

To the right of the code cell, there is a sidebar with the following text:

Create a Python program using a function to count vowels, consonants, digits, and special characters from a given string. Ignore case when checking vowels. Display output neatly using print statements like: Vowels = Consonants = Digits = Special Characters = Use input string: "AI Lab 2026!"

Below this text are two buttons: "Explain the difference between 'count_all_chars' and 'count_chars_in_string'" and "Test count_all_chars".

At the bottom of the sidebar, there is a text input field asking "What can I help you build?" and a button labeled "Gemini 2.5 Flash".

Explanation :

In this task, I analyzed a given string and counted vowels, consonants, and digits separately. This helps in understanding string processing and character classification in Python. I used conditions like `isalpha()` and `isdigit()` to differentiate letters and numbers. The output clearly shows how many vowels, consonants, and digits are present in the input text.

Task 3 : Generate a palindrome-checking function using Gemini and Copilot, then

compare the results.

Prompt Used : Write a Python function to check whether a string is palindrome. Ignore case. Show sample input and output.

The screenshot shows the AI Assisted Coding interface. On the left, there's a file tree with 'cursor1.py' selected. The main area displays the following Python code:

```
cursor1.py > ...
def is_palindrome(s):
    """
    Check if a string is a palindrome, ignoring case.

    Args:
        s (str): The string to check

    Returns:
        bool: True if the string is a palindrome, False otherwise
    """
    # Convert to Lowercase and remove spaces for comparison
    s_clean = s.lower().replace(" ", "")
    # Compare with its reverse
    return s_clean == s_clean[::-1]

# Sample inputs and outputs
if __name__ == "__main__":
    print("racecar")
    print("RaceCar")
    print("A man a plan a canal Panama")
    print("hello")
```

To the right, a sidebar titled 'Python palindrome string check' provides instructions: 'Write a Python function to check whether a string is palindrome. Ignore case. Show sample input and output.' Below it, a terminal window shows the command '\$ cd "C:/Users/hkmth/OneDrive/Desktop/AI Assisted coding" && python cursor1.py' and the output of the function for various inputs. A status bar at the bottom indicates 'Cursor Tab 1 In 43, Col 1 Spaces: 4 UTF-8'.

The screenshot shows a VS Code interface with the following details:

- Code Editor:** The file `cursor1.py` is open, containing Python code to check if strings are palindromes. It includes a list of test cases and prints the results.
- Terminal:** The terminal shows the command `PS C:\Users\hkmth\OneDrive\Desktop\AI Assisted coding & C:/Users/hkmth/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/hkmth/OneDrive/Desktop/AI Assisted coding(cursor1.py")` being run, followed by the output of the program.
- Output Panel:** Shows the execution results:
 - Input: 'racecar' Output: True - ✓ Palindrome
 - Input: 'RaceCar' Output: True - ✓ Palindrome
 - Input: 'A man a plan a canal Panama' Output: True - ✓ Palindrome
 - Input: 'hello' Output: False - X Not a palindrome
- Right Panel:** A sidebar titled "Python palindrome string check" provides a task prompt: "Write a Python function to check whether a string is palindrome. Ignore case. Show sample input and output." Below it, the generated code is shown in a code editor window, along with a note: "Check if a string is a palindrome, ign".
- Bottom Right:** A "Waiting for Approval..." status bar with "Stop" and "Review" buttons, and a "Reject, suggest, follow up?" dialog.

Explanation :

In this task, I used Cursor AI to generate a Python program that checks whether a given string is a palindrome. I entered the prompt in the Cursor AI chat panel and the tool generated the required function automatically. The code was executed in the terminal and tested with multiple inputs such as “racecar” and “hello”. Cursor AI helped in quickly generating code inside the editor and verifying output easily.

Task 4 : Ask Gemini to explain a Python function (prime check OR palindrome check) line by line.

Prompt Used : Explain the above prime check code

AI Assistant CodingAss-2.2.pynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```

def is_prime(number):
    """
    Checks if a given number is a prime number.

    A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

    Args:
        number (int): The integer to be checked.

    Returns:
        bool: True if the number is prime, False otherwise.

    ...
    if number < 1:
        return False # Numbers less than or equal to 1 are not prime
    if number == 2:
        return True # 2 is the only even prime number
    if number % 2 == 0:
        return False # Other even numbers are not prime

    # Check for divisors from 3 up to the square root of the number
    # We only need to check odd numbers as divisors
    i = 3
    while i * i <= number:
        if number % i == 0:
            return False # Found a divisor, so it's not prime
        i += 2

    return True # No divisors found, so it's prime

# Get input from the user
while True:
    try:
        user_input = int(input("Enter an integer to check if it's prime: "))
        break
    except ValueError:
        print("Invalid input. Please enter an integer.")

# Check if the number is prime and print the result
if is_prime(user_input):
    print(f"{user_input} is a prime number.")
else:
    print(f"{user_input} is not a prime number.")

Enter an integer to check if it's prime: 5
5 is a prime number.

```

Gemini

is a natural number greater than 1 that has no positive divisors other than 1 and itself. Args: number (int): The integer to be checked. Returns: bool: True if the number is prime, False otherwise. ... if number < 1: return False # Numbers less than or equal to 1 are not prime if number == 2: return True # 2 is the only even prime number if number % 2 == 0: return False # Other even numbers are not prime # Check for divisors from 3 up to the square root of the number # We only need to check odd numbers as divisors i = 3 while i * i <= number: if number % i == 0: return False # Found a divisor, so it's not prime i += 2 return True # No divisors found, so it's prime # Get input from the user # Check if the number is prime and print the result if is_prime(user_input): print(f'{user_input} is a prime number.') else: print(f'{user_input} is not a prime number.') You're asking for a line-by-line explanation of the user input and output display part of the code. Here it is:

```

# Get input from the user
while True:
    try:
        user_input = int(input("Enter an integer to check if it's prime: "))
        break
    except ValueError:
        print("Invalid input. Please enter an integer.")

# Check if the number is prime and print the result
if is_prime(user_input):
    print(f'{user_input} is a prime number.')
else:
    print(f'{user_input} is not a prime number.')

```

Line-by-line Explanation of the User Interaction Part:

- # Get input from the user
 - This is a comment indicating the purpose of the following code block: to get input from the user.
2. while True:
 - Accept X Cancel

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Gemini

3. `try:`

- This begins a `try` block. Code inside this block is monitored for potential errors (exceptions).

4. `user_input = int(input("Enter an integer to check if it's prime: "))`

- `input("Enter an integer to check if it's prime: ")`: This function displays the prompt string "Enter an integer to check if it's prime: " to the user and waits for them to type something and press Enter.
- `int(...)`: This attempts to convert the string input received from the user into an integer. If the user types something that cannot be converted to an integer (e.g., text, a decimal number), a `ValueError` will occur.
- `user_input = ...`: The successfully converted integer value is stored in the variable `user_input`.

5. `break`

- If the `int()` conversion is successful (i.e., no `ValueError` occurred), this `break` statement is executed, which immediately exits the `while True` loop.

6. `except ValueError:`

- This is an `except` block that will be executed *only if* a `ValueError` occurs within the preceding `try` block. This specifically catches the error that happens if `int()` fails.

7. `print("Invalid input. Please enter an integer.")`

- If a `ValueError` occurs (meaning the user didn't enter a valid integer), this line prints an error message to guide the user. After this, the `while True` loop will repeat, prompting the user for input again.

8. `# Check if the number is prime and print the result`

- Another comment, indicating the purpose of the subsequent code.

9. `if is_prime(user_input):`

- This calls the `is_prime` function (which was defined earlier in your code) with the `user_input` as its argument.

✓ Accept X Cancel

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Gemini

executed, which immediately exits the `while True` loop.

```
6. except ValueError:
    o This is an except block that will be executed only if a ValueError occurs within the preceding try block. This specifically catches the error that happens if int() fails.
7. print("Invalid input. Please enter an integer.")
    o If a ValueError occurs (meaning the user didn't enter a valid integer), this line prints an error message to guide the user. After this, the while True loop will repeat, prompting the user for input again.
8. # Check if the number is prime and print the result
    o Another comment, indicating the purpose of the subsequent code.
9. if is_prime(user_input):
    o This calls the is_prime function (which was defined earlier in your code) with the user_input as its argument.
    o The if statement checks the boolean value returned by is_prime(). If is_prime() returns True, the code inside the if block is executed.
10. print(f"{user_input} is a prime number.")
    o If is_prime(user_input) returned True, this line prints a formatted string indicating that the entered number is prime.
11. else:
    o If is_prime(user_input) returned False, the code inside the else block is executed.
12. print(f"{user_input} is not a prime number.")
    o If is_prime(user_input) returned False, this line prints a formatted string indicating that the entered number is not prime.
```

↶ ↷

✓ Accept X Cancel

What can I help you build?

+ Gemini 2.5 Flash ▾ ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Explanation :

In this task, I used Gemini to explain the generated code line-by-line in a simple way. This helped me understand what each line does and how the logic works internally. Gemini also explained why checking divisors up to \sqrt{n} improves performance. By reading the explanation, I learned how time complexity is reduced and how code can be optimized for efficiency.