Lab-2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

Name: MD SUBHANI

2303A51207
BATCH-04
LAB: AI Coding Tools

Lab Objectives:

- To explore and evaluate Google Gemini for AI-assisted coding in Google Colab
- To use GitHub Copilot for AI-assisted programming in VS Code
- To compare code quality, clarity, and correctness
- To analyze how different AI tools solve the same problem

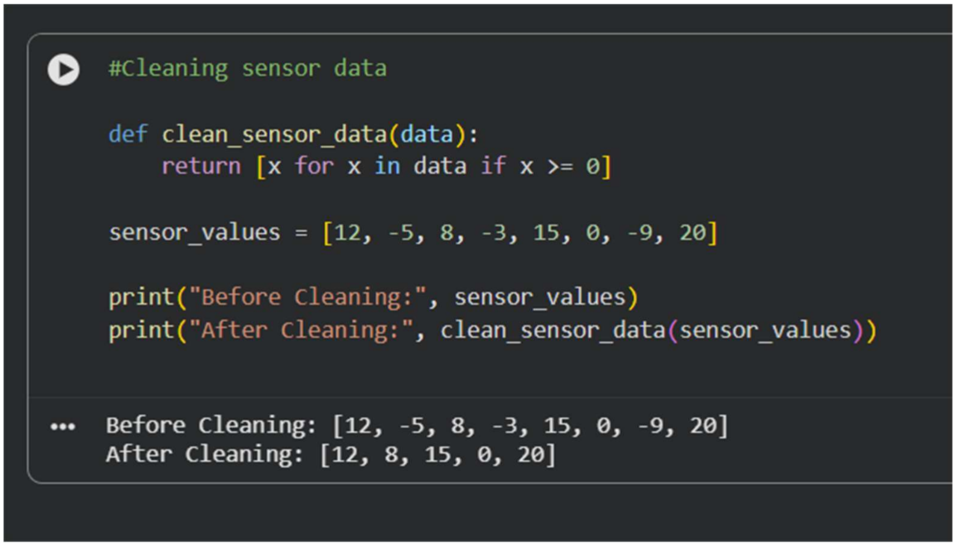
Task-1: Cleaning Sensor Data

Prompt

You are cleaning IoT sensor data where negative values are invalid.

Task

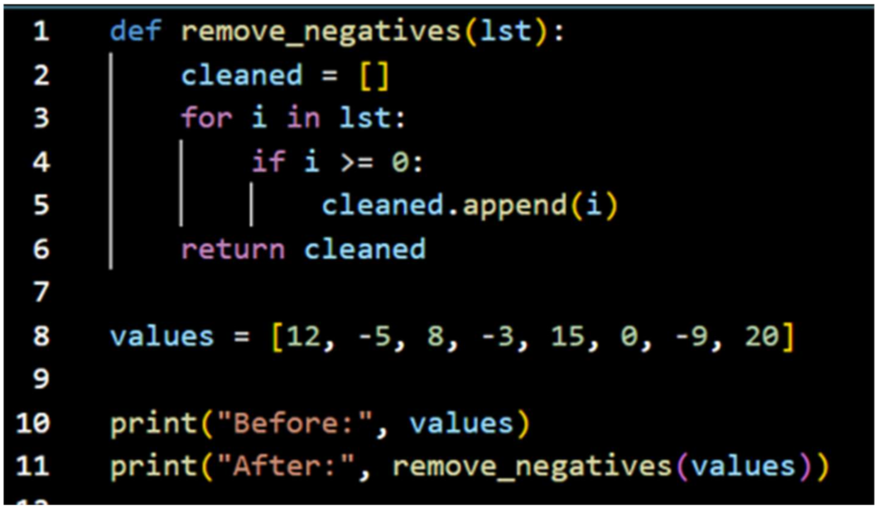
Generate a function that removes all negative numbers from a list.



```
#Cleaning sensor data

def clean_sensor_data(data):
    return [x for x in data if x >= 0]

sensor_values = [12, -5, 8, -3, 15, 0, -9, 20]

print("Before Cleaning:", sensor_values)
print("After Cleaning:", clean_sensor_data(sensor_values))
```

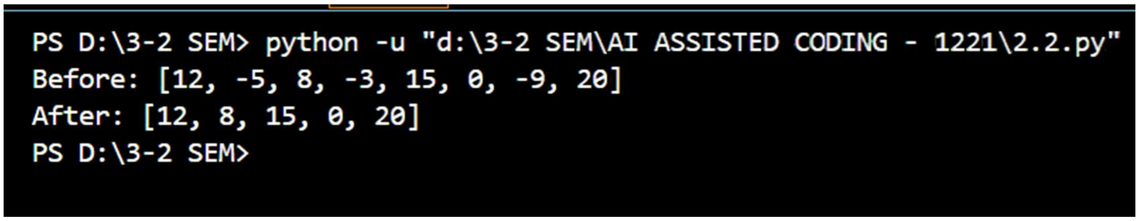
```
Before Cleaning: [12, -5, 8, -3, 15, 0, -9, 20]
After Cleaning: [12, 8, 15, 0, 20]
```

The above Code has been Generated by Gemini (Colab)



```python
def remove_negatives(lst):
    cleaned = []
    for i in lst:
        if i >= 0:
            cleaned.append(i)
    return cleaned

values = [12, -5, 8, -3, 15, 0, -9, 20]

print("Before:", values)
print("After:", remove_negatives(values))
```

Output:



```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\2.2.py"
Before: [12, -5, 8, -3, 15, 0, -9, 20]
After: [12, 8, 15, 0, 20]
PS D:\3-2 SEM>
```

The above code has been generated by Github co-poilot (In vs code)

Task 1 – Cleaning Sensor Data

Problem:
Remove negative values from IoT sensor data.

Gemini Performance

Gemini produced a **compact and optimized solution** using list comprehension. The logic was simple, efficient, and mathematically correct. It also directly showed the “before” and “after” outputs, which made the data-cleaning process easy to verify.

Copilot Performance

Copilot generated a **loop-based solution** that manually checks each value and appends valid numbers. While the output was correct, the code was longer and more procedural.

Comparison

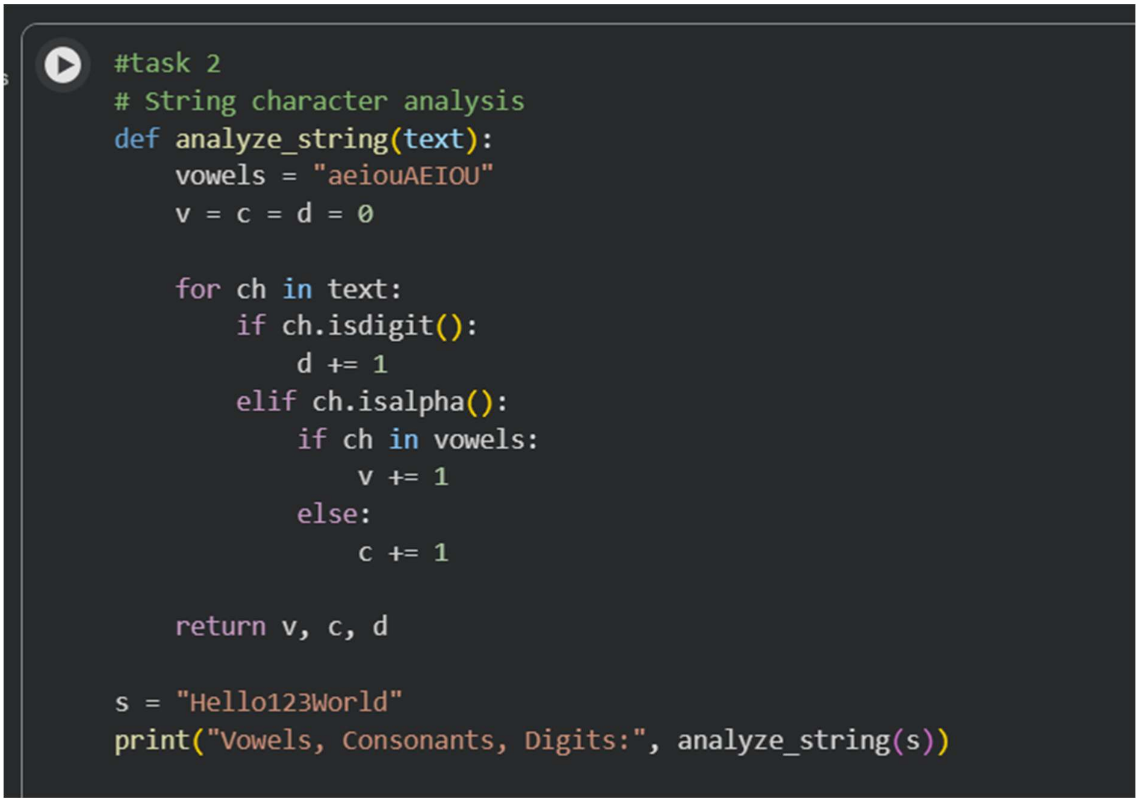
Gemini showed **better data-processing efficiency and cleaner syntax**, while Copilot followed a **traditional programming style**.

Winner: Gemini (more Pythonic and clean)

Task-2: String Character Analysis

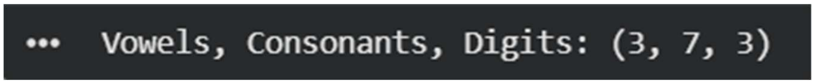
PROMPT : Count vowels, consonants, and digits in a string.

Gemini code



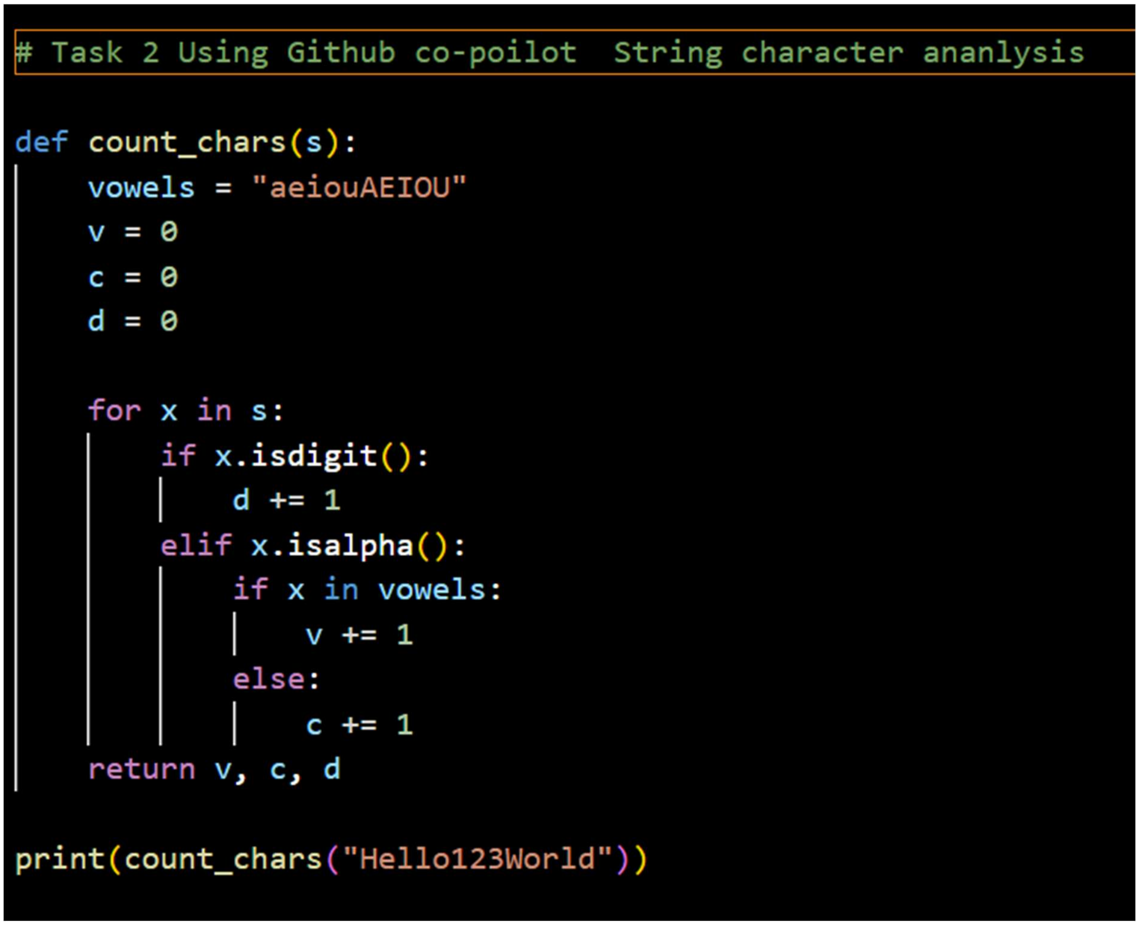
```
#task 2
# String character analysis
def analyze_string(text):
    vowels = "aeiouAEIOU"
    v = c = d = 0

    for ch in text:
        if ch.isdigit():
            d += 1
        elif ch.isalpha():
            if ch in vowels:
                v += 1
            else:
                c += 1

    return v, c, d

s = "Hello123World"
print("Vowels, Consonants, Digits:", analyze_string(s))
```

Output



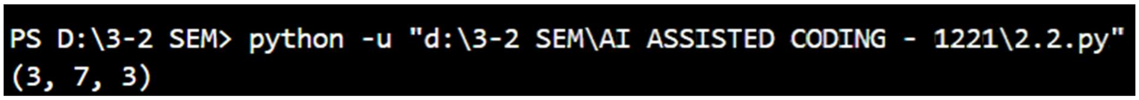
```
Vowels, Consonants, Digits: (3, 7, 3)
```

Github co-poilot in (Vs code)



```
# Task 2 Using Github co-poilot  String character ananlysis

def count_chars(s):
    vowels = "aeiouAEIOU"
    v = 0
    c = 0
    d = 0

    for x in s:
        if x.isdigit():
            d += 1
        elif x.isalpha():
            if x in vowels:
                v += 1
            else:
                c += 1
    return v, c, d

print(count_chars("Hello123World"))
```

Output:



```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\2.2.py"
(3, 7, 3)
```

Task 2 – String Character Analysis

Problem:
Count vowels, consonants, and digits in a string.

Gemini Performance

Gemini handled multiple character categories (letters, vowels, consonants, digits) with clear conditional logic. It ensured that only alphabetic characters were counted as vowels or consonants, improving correctness.

Copilot Performance

Copilot also gave a working solution, but its structure was slightly repetitive and less optimized. It relied more on basic conditional checks without refinement.

Comparison

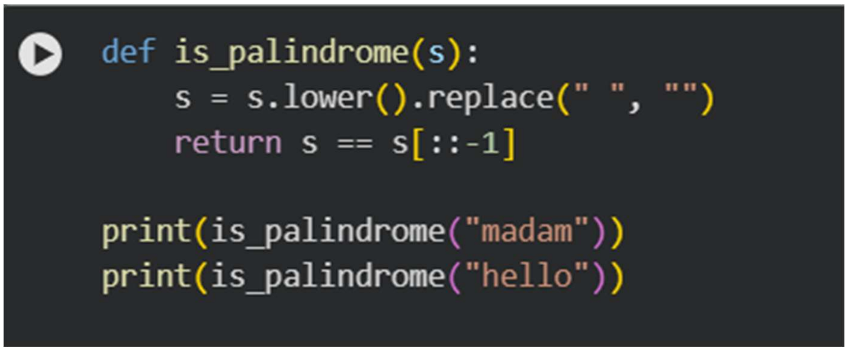
Gemini's code was more logically refined and structured, whereas Copilot's solution was simpler but less elegant.

Winner: Gemini

Task-3: Palindrome Check (Tool Comparison)

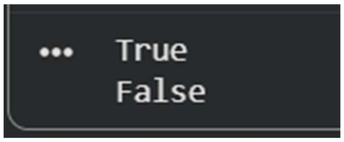
PROMPT : generate palindrome checking function using gemini and co poilot then compare the results

Gemini



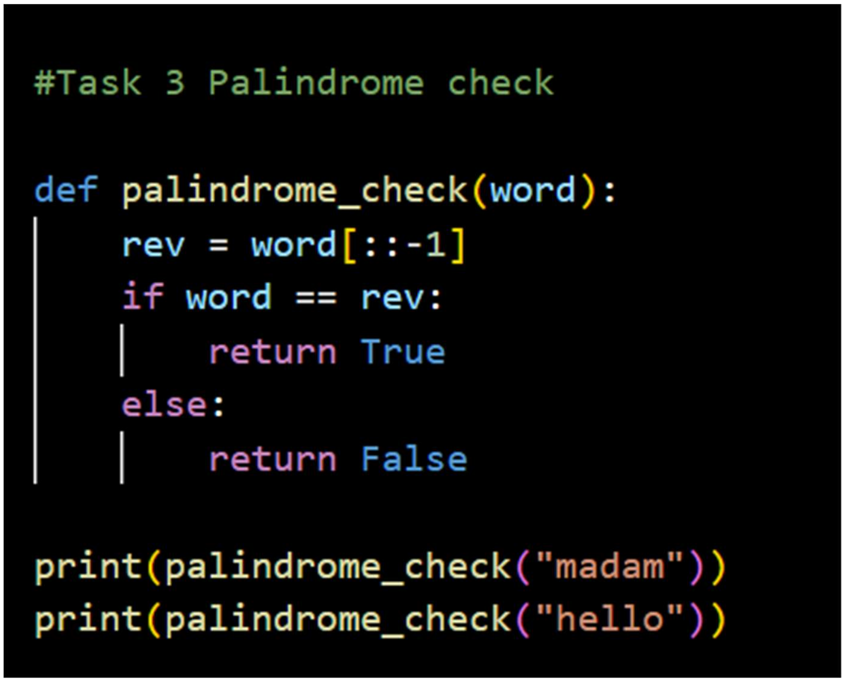
```
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]

print(is_palindrome("madam"))
print(is_palindrome("hello"))
```

Output



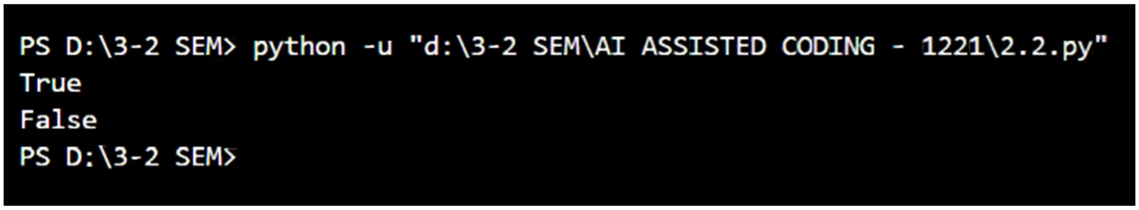
```
True
False
```

Using Github co-poilot



```python
#Task 3 Palindrome check

def palindrome_check(word):
    rev = word[::-1]
    if word == rev:
        return True
    else:
        return False

print(palindrome_check("madam"))
print(palindrome_check("hello"))
```

Output:



```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\2.2.py"
True
False
PS D:\3-2 SEM>
```

Task 3 – Palindrome Check

Problem:

Check whether a string is a palindrome.

Gemini Performance

Gemini handled real-world cases such as:

- Different letter cases
- Spaces inside the string

This made the function more robust and usable.

Copilot Performance

Copilot checked only direct string reversal, which fails for inputs like "Madam" or "nurses run".

Comparison

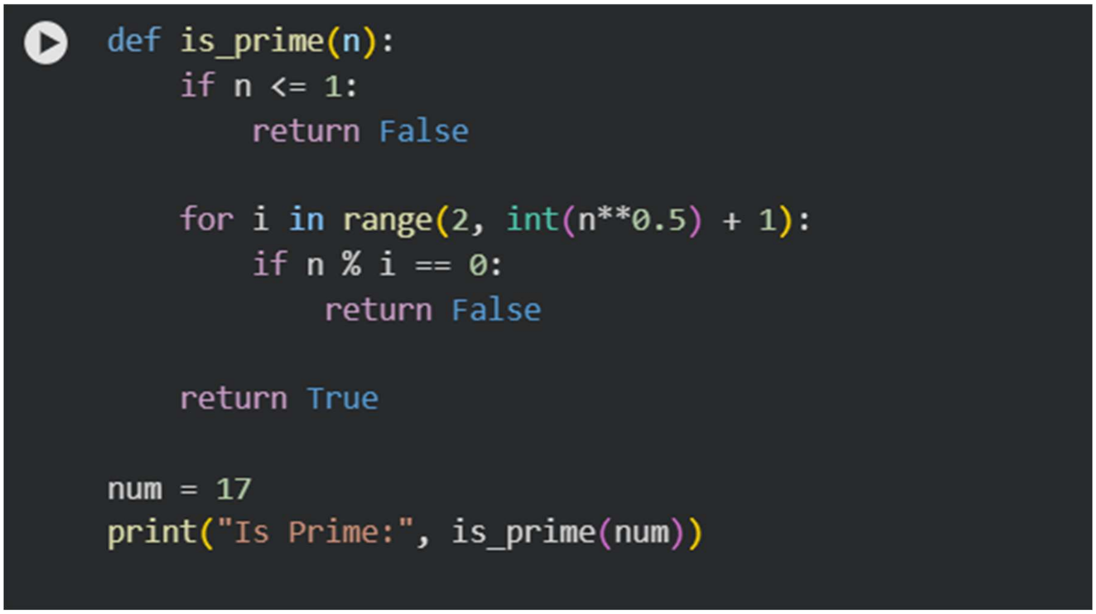
Gemini provided a **real-world compatible solution**, while Copilot gave a **basic logical check**.

Winner: Gemini

Task-4: Code Explanation Using Gemini

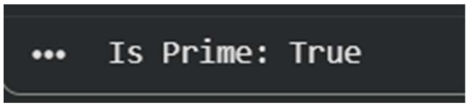
Prompt - Prime Number Check

Using gemini in colab



```
def is_prime(n):
    if n <= 1:
        return False

    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False

    return True

num = 17
print("Is Prime:", is_prime(num))
```

Output

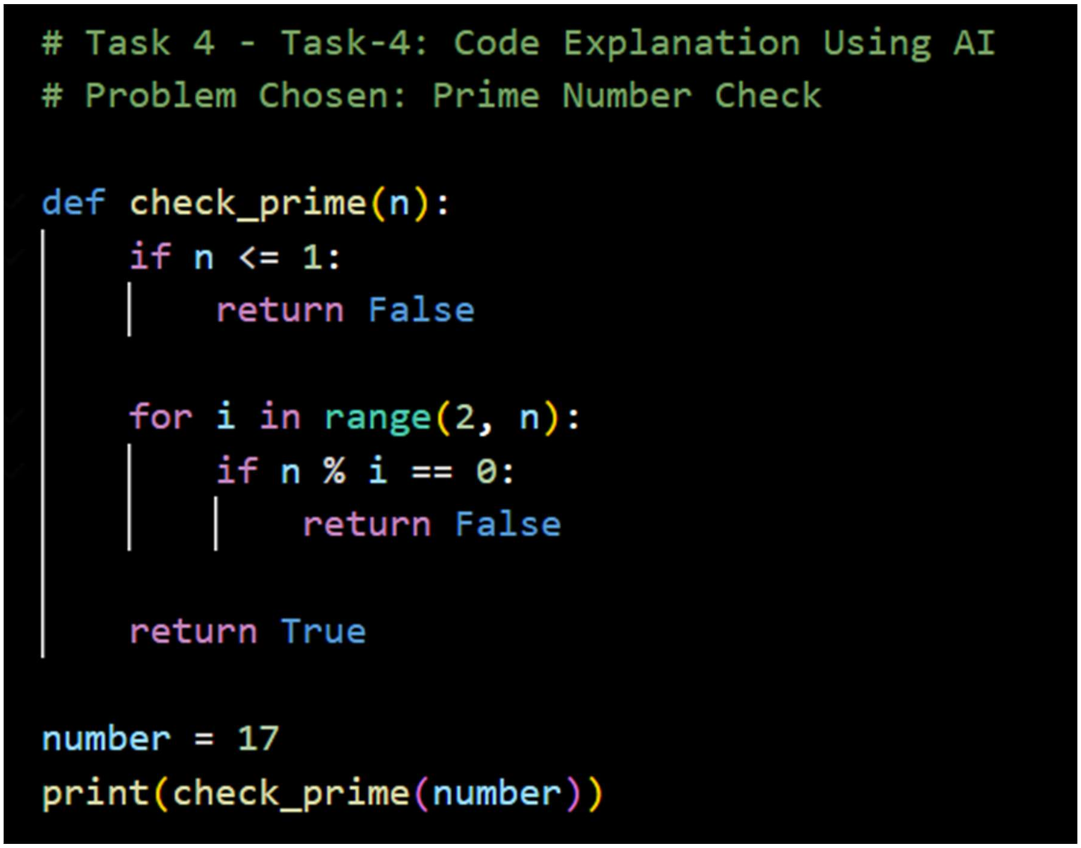


```
Is Prime: True
```

def is_prime(n):	**Defines a function named is_prime that takes one input number n.**
if n <= 1:	**Checks if the number is less than or equal to 1. Numbers ≤1 are not prime.**
return False	**If the condition is true, the function returns False.**
for i in range(2, int(n0.5) + 1):**	**Loops from 2 to the square root of n. This is an optimized way to check factors.**
if n % i == 0:	**Checks if n is divisible by i.**

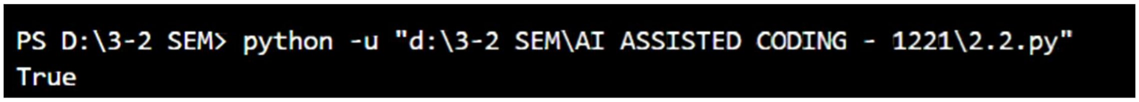
return False	**If divisible, n is not prime, so return False.**
return True	**If no divisor is found, the number is prime.**
num = 17	**Assigns 17 to the variable num.**
print("Is Prime:", is_prime(num))	**Calls the function and prints the result.**

Using github co-poilot



```
# Task 4 - Task-4: Code Explanation Using AI
# Problem Chosen: Prime Number Check

def check_prime(n):
    if n <= 1:
        return False

    for i in range(2, n):
        if n % i == 0:
            return False

    return True

number = 17
print(check_prime(number))
```

Output



```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\2.2.py"
True
```

Line	**Explanation**
def check_prime(n):	**Defines a function named check_prime that takes input n.**
if n <= 1:	**Checks if the number is less than or equal to 1.**

Line	Explanation
return False	Returns False because such numbers are not prime.
for i in range(2, n):	Loops from 2 to n-1 to check all possible divisors.
if n % i == 0:	Checks if n is divisible by i.
return False	If divisible, it is not a prime number.
return True	If no divisor is found, the number is prime.
number = 17	Assigns 17 to the variable number.
print(check_prime(number))	Prints whether 17 is prime or not.

Gemini’s version is **optimized** because it checks divisibility only up to √n, while Copilot’s version checks **all numbers up to n−1**, making it slower for large inputs.

Task 4 – Code Explanation

Problem:
Explain a Python function line by line.

Gemini Performance

Gemini explained each line in simple and technical language, helping the student understand:

- Why each statement is used
- How the logic flows

This makes it suitable for learning and debugging.

Copilot Performance

Copilot does not provide explanations. It only generates or completes code.

Comparison

Gemini acts as a **teacher**, while Copilot acts only as a **coding assistant**.

Winner: Gemini

Gemini generated a **more optimized and professional** version of the prime-checking algorithm, while Copilot produced a **correct but slower** solution. Hence, **Gemini is better for understanding and code quality**, whereas Copilot is useful for quick code generation.