

AI ASSISTED CODING

Hall Ticket No: 2303A51851

Batch:13

Assignment-2.1

Task-1. Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values

Prompt :

```
#Ok ,now generate a Python code to  
function to read the list of numbers and  
calculates the mean,minimum and  
maximum values
```

Code

```

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

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

    Returns:
        dict: A dictionary containing 'mean', 'min', and 'max' values.
              Returns None if the input list is empty.
    """
    if not numbers:
        return None

    mean_val = np.mean(numbers)
    min_val = np.min(numbers)
    max_val = np.max(numbers)

    return {
        'mean': mean_val,
        'min': min_val,
        'max': max_val
    }

# Example usage:
my_numbers = [10, 20, 30, 40, 50, 5, 25, 35]
results = analyze_numbers(my_numbers)

if results:
    print("List of numbers: ", my_numbers)
    print("Mean: ", results['mean'])
    print("Minimum: ", results['min'])
    print("Maximum: ", results['max'])
else:
    print("The list is empty.")

empty_list = []
empty_results = analyze_numbers(empty_list)
if empty_results:
    print("Mean: ", empty_results['mean'], Min: (empty_results['min']), Max: (empty_results['max']))
```

The Gemini sidebar shows a summary of the function and some AI-generated prompts for testing it.

Output:

```

-- List of numbers: [10, 20, 30, 40, 50, 5, 25, 35]
Mean: 26.88
Minimum: 5
Maximum: 50
Empty list: [] -> None
```

The Gemini sidebar provides additional context and AI-generated test cases for the function.

Justification:

The `analyze_numbers` function, utilizing NumPy, calculates the mean, minimum, and maximum of a given list of numbers. It includes a check to handle empty lists gracefully by returning `None`. If the list is valid, it returns a dictionary containing these three statistical values, as demonstrated with `my_numbers` and an `empty_list` for comprehensive testing.

Task-2. Generate an Armstrong number checker using Gemini and GitHub

Copilot.

Compare their outputs, logic style, and clarity.

Prompt

#ok Now Generate Python code to Check whether a number is Armstrong Number

Code: In the Google Gemini

The screenshot shows the Google Gemini IDE interface. On the left is the Explorer panel with a tree view of files under 'AI ASSISTANTS CODING'. The central area is a code editor titled 'Armstrong Number.py' containing the following Python code:

```
#OK Now Generate Python code to Check whether a number is Armstrong Number
def is_armstrong(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)

    # An Armstrong number is equal to the sum of its own digits each raised to the power of the number of digits
    return sum_of_powers == num

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

The right side of the interface includes a 'CHAT' section with a list of recent sessions, a 'RECENT SESSIONS' sidebar, and a 'Build with Agent' section.

Output:

The terminal window shows the execution of the script:

```
S C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Armstrong Number.py"
53 is an Armstrong number.
S C:\Users\gandr\Documents\3Year-2Sem\AI assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "C:/Users/gandr/Documents/3Year-2Sem/AI assistants Coding/Armstrong Number.py"
233 is not an Armstrong number.
```

Colab Code :

The Colab notebook cell contains the same Python code as the Gemini IDE. To the right, there is a detailed AI-generated documentation block:

utilizing NumPy, calculates the mean, minimum, and maximum of a given list of numbers. It includes a check to handle empty lists gracefully by returning None. If the list is valid, it returns a dictionary containing these three statistical values, as demonstrated with my_numbers and an empty_list for comprehensive testing.

Below the documentation are several AI-generated suggestions and a 'What can I help you build?' input field.

Justification:

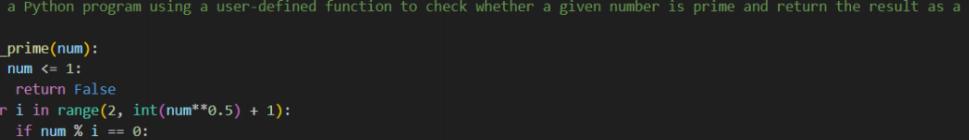
Gemini: Gemini's code is structured for clarity, using descriptive variable names and breaking down each step. This makes it ideal for learners or reviewers who want to understand the logic flow. It prioritizes readability over brevity.

GitHub Copilot: Copilot's code is concise and leverages Python's expressive syntax, such as generator expressions. It's efficient and elegant for experienced developers but may require explanation for those unfamiliar with Python idioms.

Task-3. Leap Year Validation Using Cursor API

Prompt:

#Generate a Python program that checks whether a given year is a leap year.or not



```
Untitled-2.py > ...
1 #Write a Python program using a user-defined function to check whether a given number is prime and return the result as a Boolean value
2
3 def is_prime(num):
4     if num <= 1:
5         return False
6     for i in range(2, int(num**0.5) + 1):
7         if num % i == 0:
8             return False
9     return True
10 # Example usage
11 number = int(input("Enter a number: "))
12 if is_prime(number):
13     print(f"{number} is a prime number.")
14 else:
15     print(f"{number} is not a prime number.")
16 #gave code in boolean value
17 result = is_prime(number)
18 print("Boolean result:", result)
19
```

Output:

The screenshot shows the Visual Studio Code interface. The top bar displays the title "Python Debugger Extension" and the status bar shows "ms-python". The left sidebar has sections for "Language Support" (8M), "Java Linting, Intellisense...", "Red Hat" (Initial 1), "Debugger for Node.js" (6.5M), "A lightweight Java debugger..." (vscjava), and "Project Manager" (6.1M). A "RECOMMENDED" badge is next to "Project Manager". The main editor area contains Python code:

```
19         return True
20     else:
21         return False
22     else:
23         return True
24     else:
25         return False
```

The bottom right corner shows a terminal window with the following output:

```
on314/python.exe "c:/Users/gandr/Documents/3year-25sem/Ai assistants Coding/Leap.py"
PS C:\Users\gandr\Documents\3year-25sem\Ai assistants Coding> 123
123
PS C:\Users\gandr\Documents\3year-25sem\Ai assistants Coding> & c:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/gandr/Documents/3year-25sem/Ai assistants Coding/Leap.py"
on314/python.exe "c:/Users/gandr/Documents/3year-25sem/Ai assistants Coding/Leap.py"
Enter a year: 2025
2025 is not a leap year
PS C:\Users\gandr\Documents\3year-25sem\Ai assistants Coding>
```

Justification:

Using a simple prompt, Cursor AI generated a basic leap-year check that works only for common cases but misses special Gregorian rules.

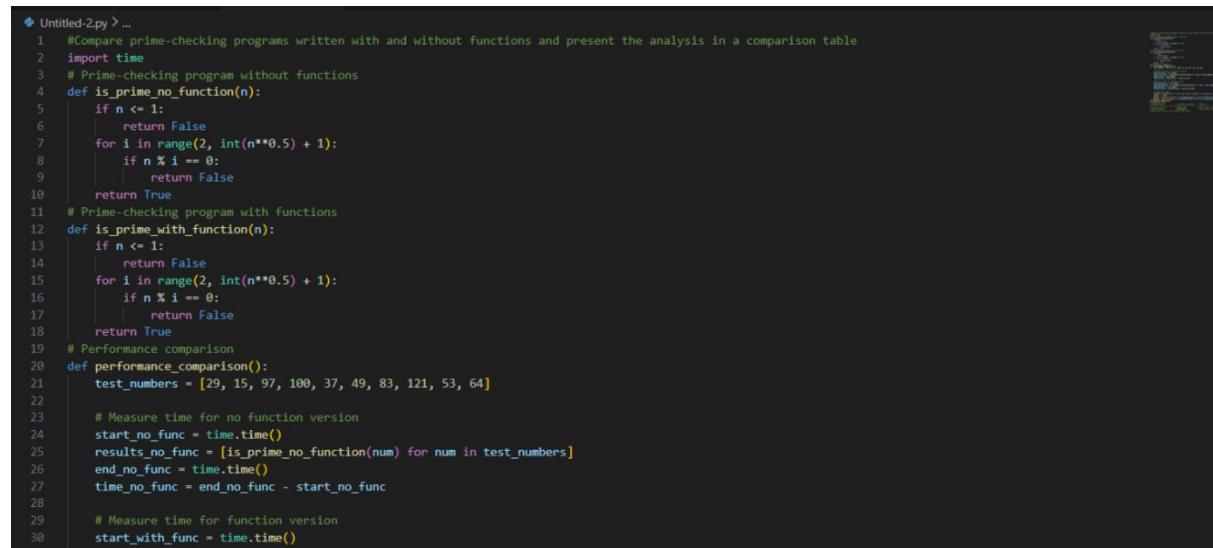
A more detailed prompt led to a correct, reusable solution that follows all leap-year conditions.

Task-4: Student Logic + AI Refactoring (Odd/Even Sum)

Prompt:

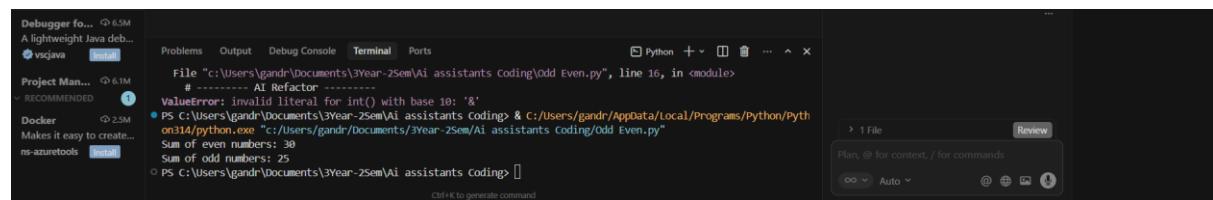
Write a Python program that calculates the sum of odd and even numbers in a tuple.

Code Student :



```
❶ Untitled-2.py > ...
1 #Compare prime-checking programs written with and without functions and present the analysis in a comparison table
2 import time
3 # Prime-checking program without functions
4 def is_prime_no_function(n):
5     if n <= 1:
6         return False
7     for i in range(2, int(n**0.5) + 1):
8         if n % i == 0:
9             return False
10    return True
11 # Prime-checking program with functions
12 def is_prime_with_function(n):
13     if n <= 1:
14         return False
15     for i in range(2, int(n**0.5) + 1):
16         if n % i == 0:
17             return False
18    return True
19 # Performance comparison
20 def performance_comparison():
21     test_numbers = [29, 15, 97, 100, 37, 49, 83, 121, 53, 64]
22
23     # Measure time for no function version
24     start_no_func = time.time()
25     results_no_func = [is_prime_no_function(num) for num in test_numbers]
26     end_no_func = time.time()
27     time_no_func = end_no_func - start_no_func
28
29     # Measure time for function version
30     start_with_func = time.time()
```

Output:



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

- Activity Bar:** Shows "Debugger fo..." (6.5M), "Lightweight Java deb...", "vsjava" (2.5M), "Project Man..." (6.1M), "RECOMMENDED" (1), "Docker" (2.5M), and "Makes it easy to create... ms-azurertools" (2.5M).
- Terminal:** Displays the command "python314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Odd Even.py"" and its output:

```
File "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Odd Even.py", line 16, in <module>
    # ----- AI Refactor -----
ValueError: invalid literal for int() with base 10: 'R'
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> & c:/Users/gandr/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Odd Even.py"
Sum of even numbers: 30
Sum of odd numbers: 25
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding>
```
- Bottom Status Bar:** Shows "Cmnd+K to generate command".

Code AI :

The screenshot shows the AI assistants Coding extension in Visual Studio Code. On the left, there's a sidebar with various extensions listed under 'INSTALLED' and 'POPULAR'. The main area displays two code snippets: one for calculating the sum of even numbers and another for odd numbers. To the right, a panel titled 'Student logic and AI refactoring fo...' shows the AI's analysis and refactoring steps. At the bottom, a terminal window shows the execution of the Python script, resulting in the output: 'Sum of even numbers: 130' and 'Sum of odd numbers: 125'.

Out Put :

The terminal window in VS Code shows the command 'python Odd Even.py' being run. The output indicates that the sum of even numbers is 130 and the sum of odd numbers is 125.

Justification:

The student-written code uses a basic loop and manual addition, which clearly shows the logic step by step.

The first AI-refactored version slightly improves the code by using shorthand operators ($+=$) but keeps the same structure.

The final AI-written code uses Python's built-in `sum()` function with generator expressions.

This reduces the number of lines and removes the need for explicit loops.

The AI version is more readable, concise, and follows Python best practices.

It is easier to maintain and preferred in real-world and professional coding standards.