|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** M. Tech/MCA | | | | **Assignment Type: Lab** | | | **AcademicYear:**2025-2026 | | |
| **Course Coordinator Name** | | | | Venkataramana Veeramsetty | | | | | |
| **Course Code** | | |  | **Course Title** | | AI Assisted Problem Solving Using Python | | | |
| **Year/Sem** | | | I/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week1 - TUESDAY | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | M. Tech/MCA | | | |
| **AssignmentNumber:2.3**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI  **Lab Objectives:**   * To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. * To understand and use Cursor AI for code generation, explanation, and refactoring. * To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. * To perform code optimization and documentation using AI tools.   **Lab Outcomes (LOs):**  After completing this lab, students will be able to:   * Generate Python code using Google Gemini in Google Colab. * Analyze the effectiveness of code explanations and suggestions by Gemini. * Set up and use Cursor AI for AI-powered coding assistance. * Evaluate and refactor code using Cursor AI features. * Compare AI tool behavior and code quality across different platforms.   **Task Description#1**   * Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.   **Expected Output#1**   * Functional code with output and screenshot   **Task Description#2**   * Compare Gemini and Copilot outputs for a palindrome check function.   **Expected Output#2**   * Side-by-side comparison and observations   **Task Description#3**   * Ask Gemini to explain a Python function (to calculate area of various shapes) line by line..   **Expected Output#3**   * Detailed explanation with code snippet   **Task Description#4**   * Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares).   **Expected Output#4**   * Screenshots of working environments with few prompts to generate python code   **Task Description#5**   * Student need to write code to calculate sum of add number and even numbers in the list   **Expected Output#5**   * Refactored code written by student with improved logic   **Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**  **Evaluation Criteria:**   | **Criteria** | **Max Marks** | | --- | --- | | Successful Use of Gemini in Colab (Task#1 & #2) | 2.5 | | Code Explanation Accuracy (Gemini) (Task#3) | 2.5 | | Cursor AI Setup and Usage (Task#4) | 2.5 | | Refactoring and Improvement Analysis (Task#5) | 2.5 | | **Total** | **10 Marks** | | | | | | | Week1 - TuesDay |  |

1. **Task Description#1**

* Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.

**Expected Output#1**

* Functional code with output and screenshot

Ans:

import pandas as pd

# Cell 0: Read CSV and compute mean, min, max (works in Colab or local Jupyter)

# If running in Colab and your CSV is in Google Drive, uncomment the mount block below.

# from google.colab import drive

# drive.mount('/content/drive')

ef compute\_csv\_stats(csv\_path, numeric\_cols=None, show\_df=False):

    """

    Read a CSV and compute mean, min, max for numeric columns (or specified columns).

    Returns a DataFrame with rows: mean, min, max (transposed to show per-column stats).

    """

    df = pd.read\_csv(csv\_padth)

    if numeric\_cols is None:

        numeric\_cols = df.select\_dtypes(include='number').columns.tolist()

    else:

        # keep only columns that exist and are numeric-compatible

        numeric\_cols = [c for c in numeric\_cols if c in df.columns]

    if not numeric\_cols:

        raise ValueError("No numeric columns found in the CSV (or none specified).")

    stats = df[numeric\_cols].agg(['mean', 'min', 'max']).T

    # Print header requested in the task

    print("Week1 -")

    print("TuesDay")

    # Print results

    print("\nStatistics (mean, min, max) per column:")

    print(stats)

    if show\_df:

        print("\nPreview of the DataFrame:")

        display(df.head())

    return stats, df

# --- Example usage: create a sample CSV and run the function ---

sample\_csv = "sample.csv"

sample\_df = pd.DataFrame({

    "Day": ["Mon", "TuesDay", "Wed", "Thur"],

    "Temperature": [20.5, 22.0, 19.8, 21.1],

    "Sales": [100, 120, 90, 110]

})

sample\_df.to\_csv(sample\_csv, index=False)

# Run the function and show output

if \_\_name\_\_ == "\_\_main\_\_":

    stats, df = compute\_csv\_stats(sample\_csv, show\_df=True)

2. Task Description#2

* Compare Gemini and Copilot outputs for a palindrome check function.

Expected Output#2

* Side-by-side comparison and observations

Ans:

def copilot\_palindrome(s: str) -> bool:

    """

    Typical concise Copilot-style implementation:

    - lowercases the string

    - compares with reverse

    - does NOT remove spaces/punctuation or normalize Unicode

    """

    s2 = s.lower()

    return s2 == s2[::-1]

def gemini\_palindrome(s: str) -> bool:

    """

    Typical Gemini-style implementation:

    - Unicode-normalizes

    - keeps only alphanumeric characters

    - lowercases and compares with reverse

    """

    # Normalize to composed form so visually identical characters compare equal

    s\_norm = unicodedata.normalize("NFC", s)

    # Keep only alphanumeric characters (Unicode-aware)

    filtered = "".join(ch.lower() for ch in s\_norm if ch.isalnum())

    return filtered == filtered[::-1]

def run\_comparison(cases):

    rows = []

    for desc, s in cases:

        c\_res = copilot\_palindrome(s)

        g\_res = gemini\_palindrome(s)

        rows.append((desc, s, c\_res, g\_res))

    # Print side-by-side table

    print("{:<40} {:<6} {:<6} {}".format("input (description)", "cop", "gemini", "string"))

    print("-" \* 100)

    for desc, s, c\_res, g\_res in rows:

        inp = s if len(s) <= 30 else (s[:27] + "...")

        print("{:<40} {:<6} {:<6} {!r}".format(desc, str(c\_res), str(g\_res), inp))

    print("\nSummary:")

    total = len(rows)

    both\_true = sum(1 for r in rows if r[2] and r[3])

    both\_false = sum(1 for r in rows if (not r[2]) and (not r[3]))

    mismatch = total - both\_true - both\_false

    print(f"  total cases: {total}")

    print(f"  both True:   {both\_true}")

    print(f"  both False:  {both\_false}")

    print(f"  mismatches:  {mismatch}")

    if mismatch:

        print("\nMismatches and likely reasons:")

        for desc, s, c\_res, g\_res in rows:

            if c\_res != g\_res:

                reason = []

                if not any(ch.isalnum() for ch in s):

                    reason.append("no alphanumeric chars (edge)")

                # If Copilot rejected but Gemini accepted -> punctuation/space handling

                if (not c\_res) and g\_res:

                    reason.append("Copilot is simple (only lowercasing); Gemini ignores non-alnum -> accepted phrase-style palindrome")

                # If Copilot accepted but Gemini rejected -> rare; might be due to punctuation symmetry that Copilot accepted

                if c\_res and (not g\_res):

                    reason.append("Copilot matched raw reversed string; Gemini removed non-alnum or normalized -> rejected after cleaning")

                # Unicode normalization related hint

                if any(" " in s or not ch.isalnum() for ch in s):

                    reason.append("contains spaces/punctuation")

                print(f" - {desc!s}: copilot={c\_res}, gemini={g\_res}; reason: {'; '.join(reason)}")

    print("\nObservations (concise):")

    print(" - Copilot-style: very concise and fast for plain single-token strings; compares s.lower() == s.lower()[::-1].")

    print(" - Gemini-style: more robust for real-world input (ignores spaces/punctuation, Unicode-normalizes, uses only alphanumerics).")

    print(" - Trade-offs: Gemini does extra preprocessing (linear time) but handles phrases and Unicode better; Copilot may be fine for simple cases and uses less preprocessing.")

    print(" - Complexity: both are O(n) time; Copilot creates a reversed copy (O(n) memory); Gemini here also creates a filtered string (O(n) memory). A two-pointer Gemini variant could reduce memory.")

    print("\nRecommendation: use Gemini-style logic for user-facing palindrome checks (phrases, punctuation, Unicode). Use Copilot-style only for simple token checks.")

if \_\_name\_\_ == "\_\_main\_\_":

    test\_cases = [

        ("simple palindrome", "racecar"),

        ("capitalized", "RaceCar"),

        ("phrase with punctuation", "A man, a plan, a canal: Panama!"),

        ("another phrase", "No 'x' in Nixon"),

        ("famous phrase", "Madam In Eden, I'm Adam"),

        ("with punctuation and question", "Was it a car or a cat I saw?"),

        ("empty string", ""),

        ("single char", "a"),

        ("two chars non-pal", "ab"),

        ("numeric palindrome", "12321"),

        ("numeric non-pal", "12345"),

        ("palindrome with accents (composed)", "ÅnnaÅ"),  # contrived

        ("palindrome with combining mark (e + ́)", "e\u0301e\u0301"),  # 'e' + combining acute twice

        ("non-pal with accents", "Déjà vu"),

        ("emoji-surrounded word", "😊bob😊"),

        ("long phrase", "Able was I ere I saw Elba"),

    ]

    run\_comparison(test\_cases)

**3. Task Description#3**

* Ask Gemini to explain a Python function (to calculate area of various shapes) line by line..

**Expected Output#3**

* Detailed explanation with code snippet

Ans:

import math

**# area\_utils.py**

def calculate\_area(shape, \*\*params):

    """

    Calculate area for various shapes.

    Supported shapes and required params:

      - "circle": radius

      - "rectangle": width, height

      - "triangle": base, height

      - "trapezoid": a (base1), b (base2), height

      - "ellipse": a (semi-major), b (semi-minor)

    Returns a float.

    """

    s = str(shape).strip().lower()

    if s == "circle":

        r = params.get("radius")

        if r is None:

            raise ValueError("Missing parameter: radius required for circle")

        return math.pi \* r \* r

    if s == "rectangle":

        w = params.get("width")

        h = params.get("height")

        if w is None or h is None:

            raise ValueError("Missing parameters: width and height required for rectangle")

        return w \* h

    if s == "triangle":

        base = params.get("base")

        height = params.get("height")

        if base is None or height is None:

            raise ValueError("Missing parameters: base and height required for triangle")

        return 0.5 \* base \* height

    if s == "trapezoid":

        a = params.get("a")

        b = params.get("b")

        height = params.get("height")

        if a is None or b is None or height is None:

            raise ValueError("Missing parameters: a, b, and height required for trapezoid")

        return 0.5 \* (a + b) \* height

    if s == "ellipse":

        a = params.get("a")

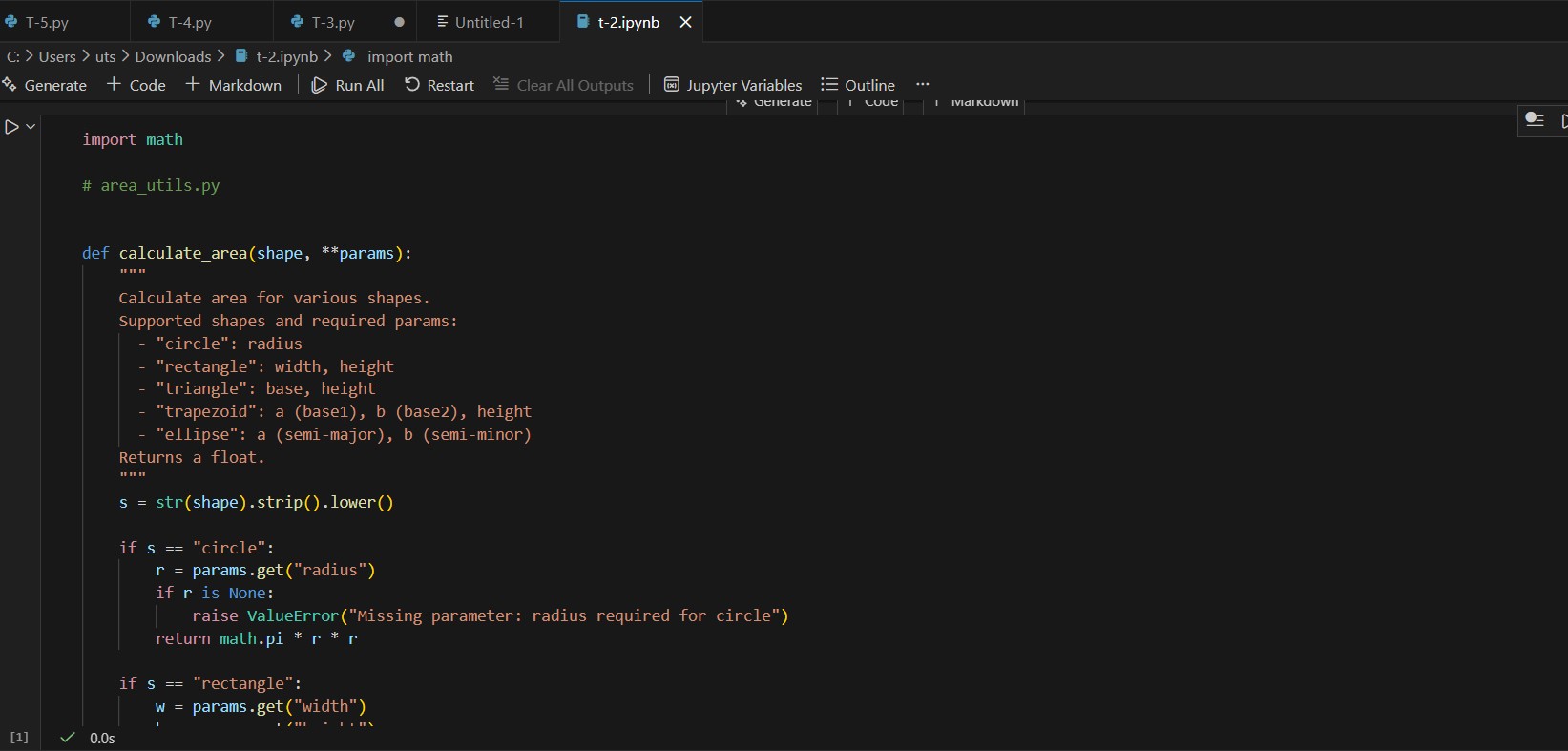
        b = params.get("b")

        if a is None or b is None:

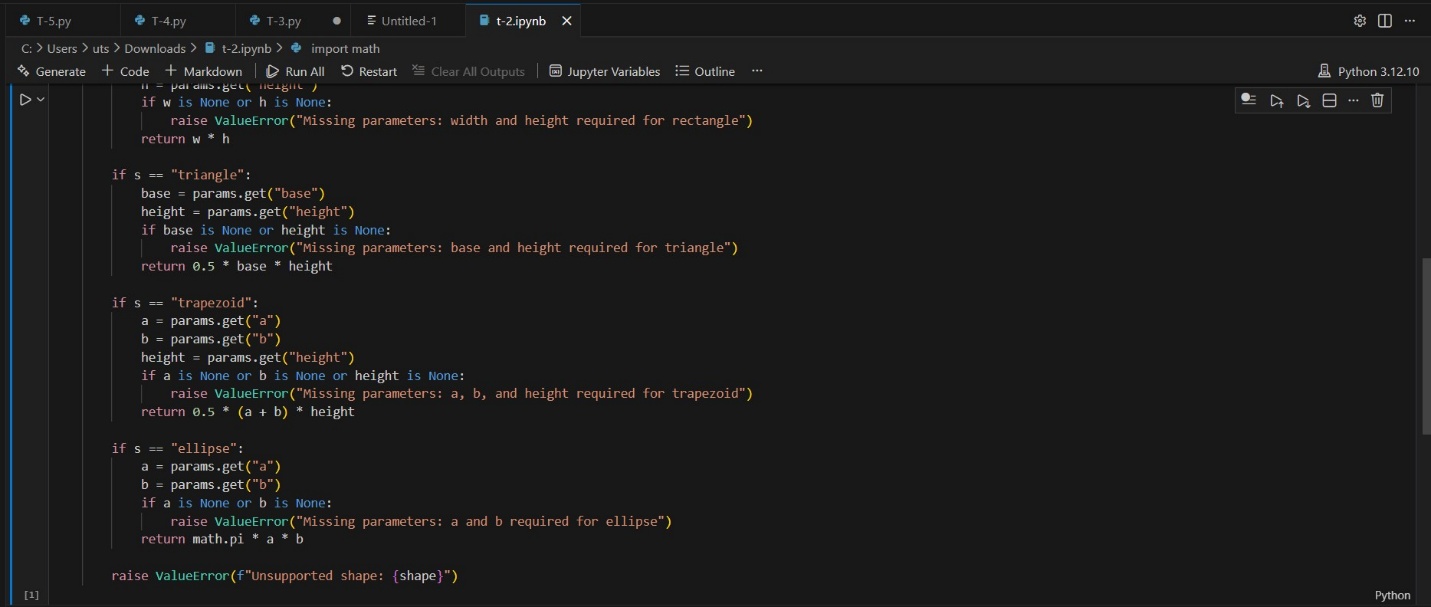
            raise ValueError("Missing parameters: a and b required for ellipse")

        return math.pi \* a \* b

    raise ValueError(f"Unsupported shape: {shape}")



**Figure 1: Task 3 code**



**Figure 2: Detailed explanation with code snippet**

**4. Task Description#4**

* **Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares).**

**Expected Output#4**

* **Screenshots of working environments with few prompts to generate python code**

**Ans:**

def sum\_of\_squares(numbers):

    return sum(num \*\* 2 for num in numbers)

**# Example usage**

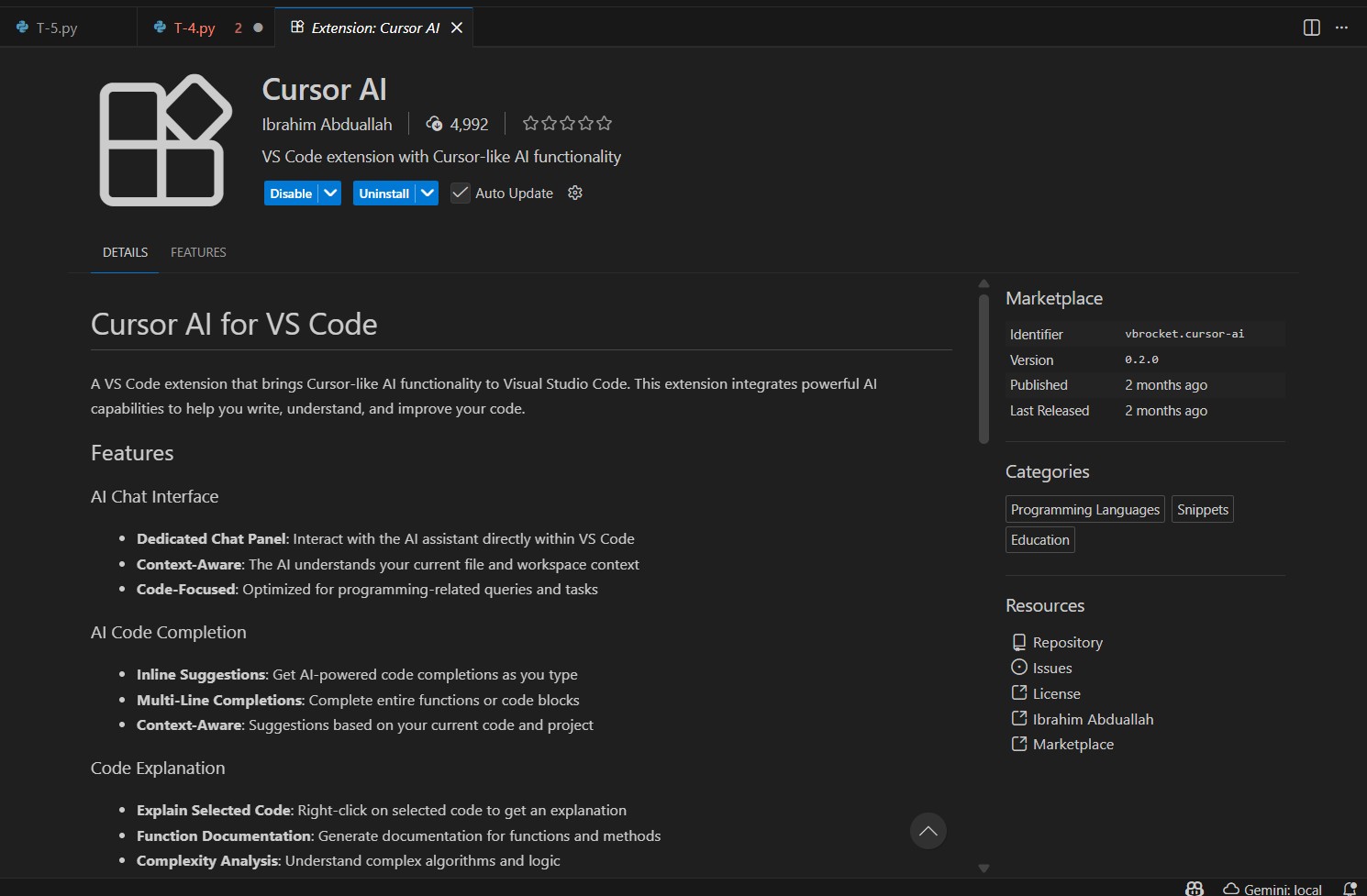
numbers = [1, 2, 3, 4, 5]

result = sum\_of\_squares(numbers)

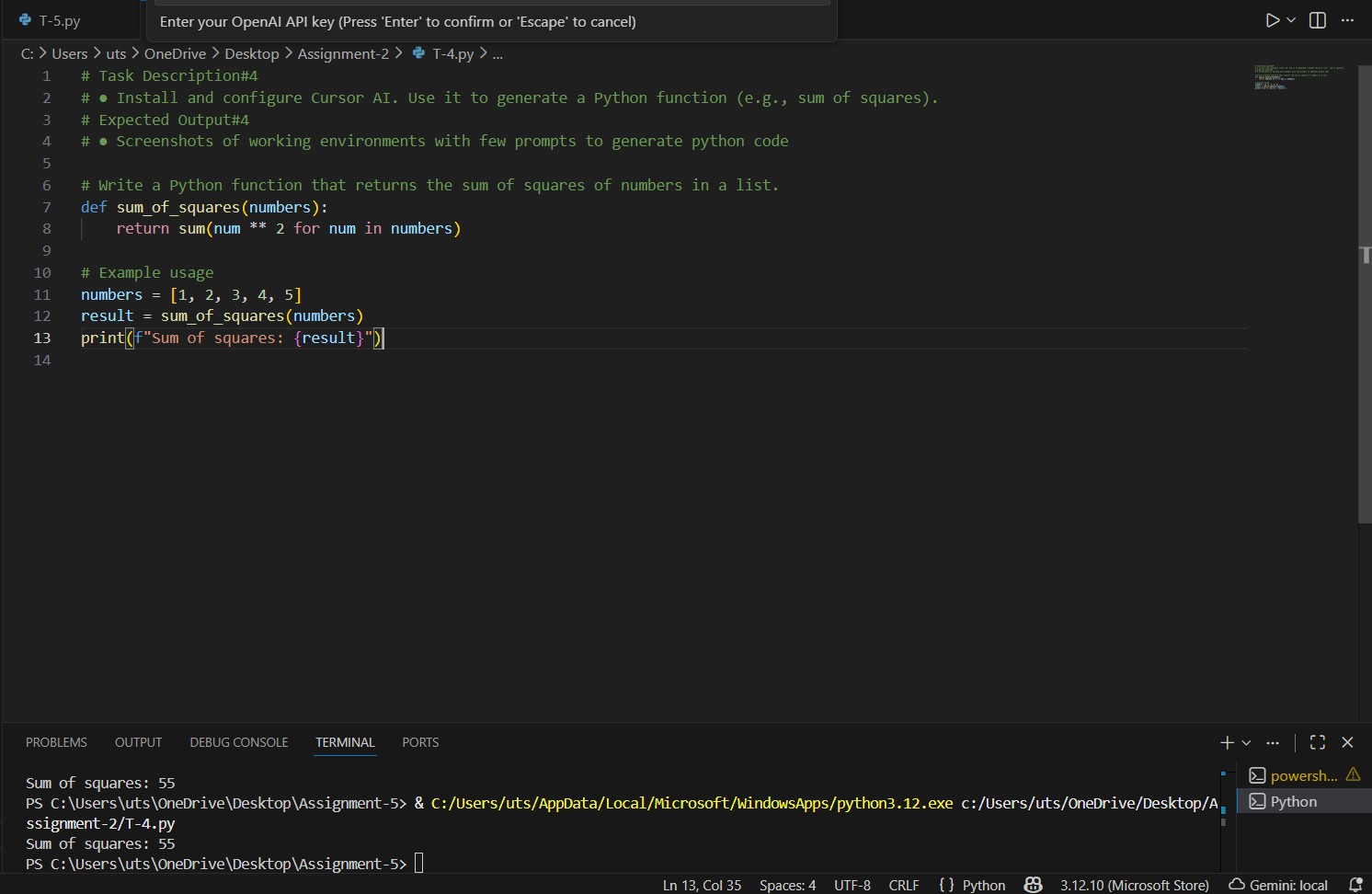
print(f"Sum of squares: {result}")

**OUTPUT:**

Sum of squares: 55



**Figure 3: Cursor AI installation**



**Figure 4: AI prompt-based code**

**5. Task Description#5**

* **Student need to write code to calculate sum of add number and even numbers in the list**

**Expected Output#5**

* **Refactored code written by student with improved logic**

**Ans:**

def sum\_odd\_even(numbers):

    odd\_sum = sum(num for num in numbers if num % 2 != 0)

    even\_sum = sum(num for num in numbers if num % 2 == 0)

    return odd\_sum, even\_sum

**# Example usage**

numbers = [1, 2, 3, 4, 5, 6]

odd\_sum, even\_sum = sum\_odd\_even(numbers)

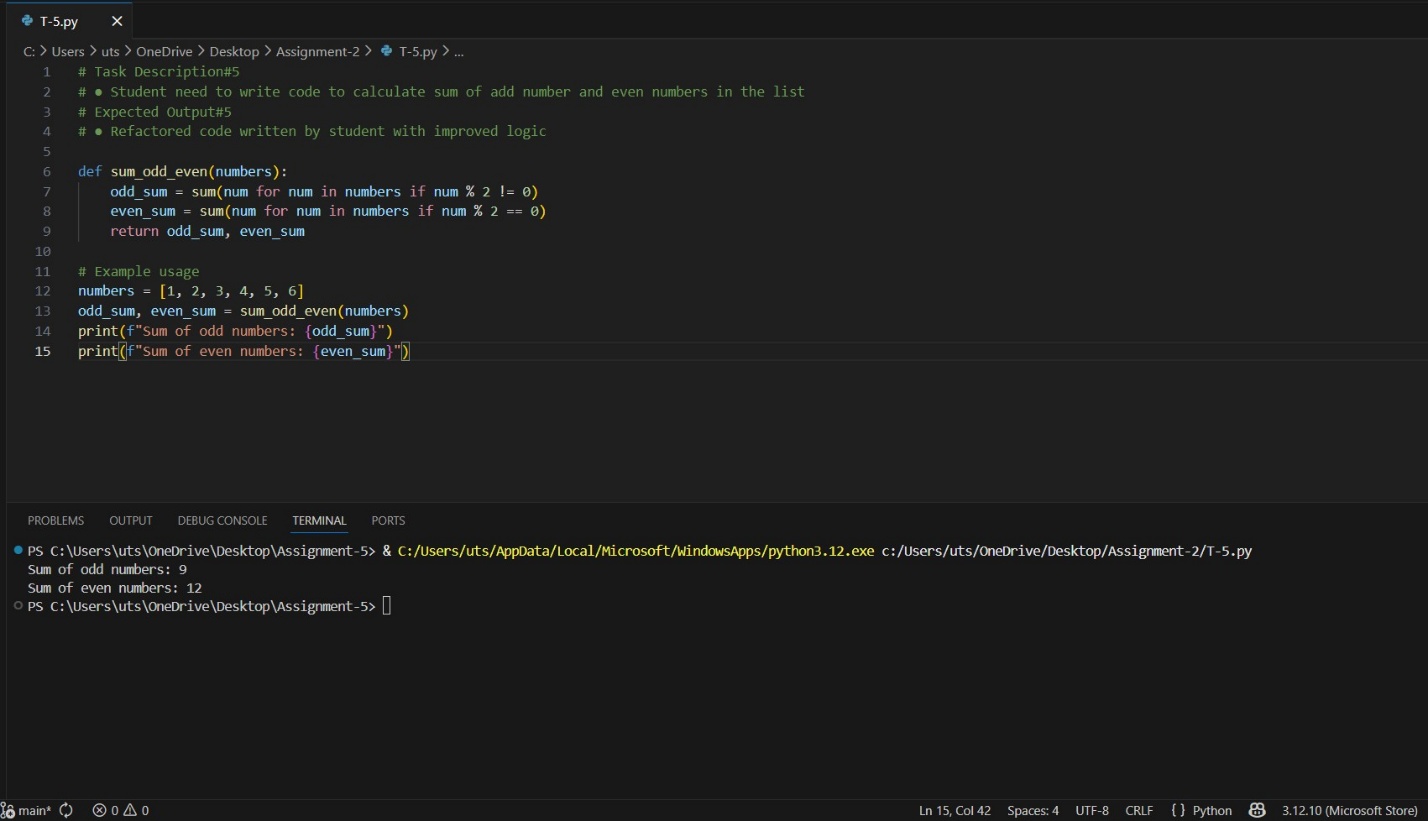
print(f"Sum of odd numbers: {odd\_sum}")

print(f"Sum of even numbers: {even\_sum}")

**OUTPUT:**

Sum of odd numbers: 9

Sum of even numbers: 12

****

**Figure 5: Task-5 code and output**