

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING		
Program Name: M. Tech/MCA		Assignment Type: Lab		
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	<p>Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and Cursor AI</p> <p>Lab Objectives:</p> <ul style="list-style-type: none"> 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. <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> 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. <p>Task Description#1</p> <ul style="list-style-type: none"> Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max. <p>Expected Output#1</p>	Week1 - TuesDay

	<ul style="list-style-type: none"> Functional code with output and screenshot <p>Task Description#2</p> <ul style="list-style-type: none"> Compare Gemini and Copilot outputs for a palindrome check function. <p>Expected Output#2</p> <ul style="list-style-type: none"> Side-by-side comparison and observations <p>Task Description#3</p> <ul style="list-style-type: none"> Ask Gemini to explain a Python function (to calculate area of various shapes) line by line.. <p>Expected Output#3</p> <ul style="list-style-type: none"> Detailed explanation with code snippet <p>Task Description#4</p> <ul style="list-style-type: none"> Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares). <p>Expected Output#4</p> <ul style="list-style-type: none"> Screenshots of working environments with few prompts to generate python code <p>Task Description#5</p> <ul style="list-style-type: none"> Student need to write code to calculate sum of odd numbers and even numbers in the list <p>Expected Output#5</p> <ul style="list-style-type: none"> Refactored code written by student with improved logic <p>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</p> <p>Evaluation Criteria:</p> <table border="1"> <thead> <tr> <th>Criteria</th><th>Max Marks</th></tr> </thead> <tbody> <tr> <td>Successful Use of Gemini in Colab (Task#1 & #2)</td><td>2.5</td></tr> <tr> <td>Code Explanation Accuracy (Gemini) (Task#3)</td><td>2.5</td></tr> <tr> <td>Cursor AI Setup and Usage (Task#4)</td><td>2.5</td></tr> <tr> <td>Refactoring and Improvement Analysis (Task#5)</td><td>2.5</td></tr> <tr> <td>Total</td><td>10 Marks</td></tr> </tbody> </table>	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	
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													

TASK DESCRIPTION -1

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

```
File C:\...\python assessment\lab 2 assessment ● import csv Untitled-2 ● + TASK 1.py X ▶ □ ...  
C: > Users > rimsha > OneDrive > Desktop > Mohammed Farnas Ali Mudabbir > LAB 2 > + TASK 1.py > ...  
● 1 import csv  
2 import statistics  
3 import os  
4  
5 # Ensure the CSV is saved inside the 'Assignment2' folder  
6 folder = "Assignment2"  
7 os.makedirs(folder, exist_ok=True) # create folder if it doesn't exist  
8 csv_path = os.path.join(folder, "data.csv")  
9  
10 # Data to write into the CSV file  
11 data = [  
12     ["Name", "Age", "Score"],  
13     ["Raj", 21, 88],  
14     ["Priya", 22, 92],  
15     ["Amit", 20, 75]  
16 ]  
17  
18 # Create and write to a CSV file inside Assignment2 folder  
19 with open(csv_path, mode="w", newline="") as file:  
20     writer = csv.writer(file)  
21     writer.writerows(data)  
22  
23 print(f"CSV file created successfully as '{csv_path}'")  
24  
25 def analyze_csv(path):  
26     """  
27         Read CSV at path and compute mean, min, max for each numeric column.  
28         Returns a dict mapping column -> {'mean':..., 'min':..., 'max':...}  
29     """
```

```
29     ....
30     with open(path, newline='') as f:
31         reader = csv.DictReader(f)
32         if not reader.fieldnames:
33             return {}
34         cols = {name: [] for name in reader.fieldnames}
35         for row in reader:
36             for name, value in row.items():
37                 try:
38                     cols[name].append(float(value))
39                 except (TypeError, ValueError):
40                     continue # ignore non-numeric values
41
42     results = {}
43     for name, values in cols.items():
44         if values:
45             results[name] = {
46                 "mean": statistics.mean(values),
47                 "min": min(values),
48                 "max": max(values),
49             }
50     return results
51
52
53 # Example usage: prints stats for numeric columns in 'Assignment2.dat'
54 if __name__ == "__main__":
55     stats = analyze_csv(csv_path)
56     for col, s in stats.items():
57         print(f'{col}: {s["mean"]}, {s["min"]}, {s["max"]}
```

```
C: > Users > rimsha > OneDrive > Desktop > Mohammed Farnas Ali Mudabbir > LAB 2 > TASK 1.py > ...
25     def analyze_csv(path):
● 43     for name, values in cols.items():
44         if values:
45             results[name] = {
46                 "mean": statistics.mean(values),
47                 "min": min(values),
48                 "max": max(values),
49             }
50     return results
51
52
53 # Example usage: prints stats for numeric columns in 'Assignment2/data.csv'
54 if __name__ == "__main__":
55     stats = analyze_csv(csv_path)
56     for col, s in stats.items():
57         print(f"{col}: mean={s['mean']}, min={s['min']}, max={s['max']}
```

Expected Output

Functional code with output and screenshot

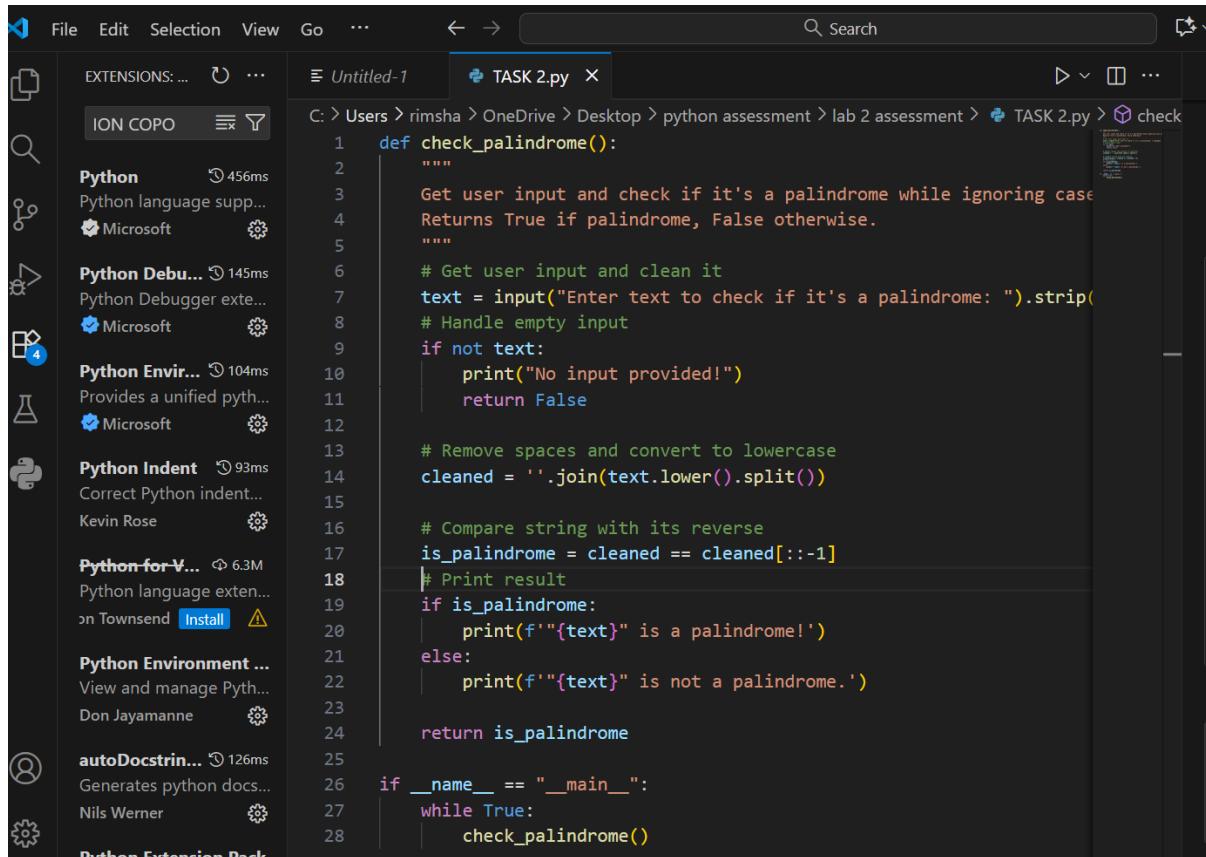
Practical output:

```
● PS C:\Users\rimsha> python -u "c:\Users\rimsha\OneDrive\Desktop\Mohammed Farnas Ali Mudabbir\LAB 2\task 1.py"
✓ CSV file created successfully as 'Assignment2\data.csv'
Age: mean=21.0, min=20.0, max=22.0
Score: mean=85.0, min=75.0, max=92.0
❖ PS C:\Users\rimsha>
```

TASK DESCRIPTION -2

Compare Gemini and Copilot outputs for a palindrome check function.

Prompt: write a user input palindrome function



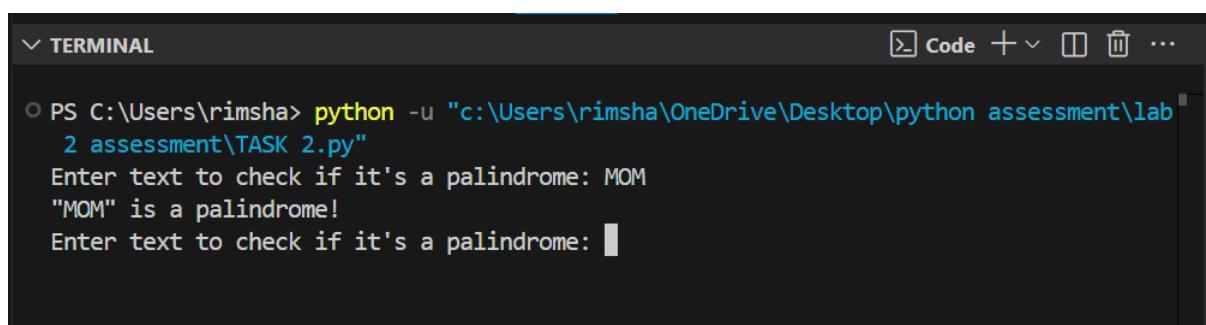
The screenshot shows the Visual Studio Code interface. On the left is the Extensions sidebar with various Python-related extensions listed. The main workspace shows an untitled file named "TASK 2.py". The code in the file is as follows:

```
1 def check_palindrome():
2     """
3         Get user input and check if it's a palindrome while ignoring case
4         Returns True if palindrome, False otherwise.
5     """
6
7     # Get user input and clean it
8     text = input("Enter text to check if it's a palindrome: ").strip()
9     # Handle empty input
10    if not text:
11        print("No input provided!")
12        return False
13
14    # Remove spaces and convert to lowercase
15    cleaned = ''.join(text.lower().split())
16
17    # Compare string with its reverse
18    is_palindrome = cleaned == cleaned[::-1]
19    # Print result
20    if is_palindrome:
21        print(f'{text} is a palindrome!')
22    else:
23        print(f'{text} is not a palindrome.')
24
25    return is_palindrome
26
27 if __name__ == "__main__":
28     while True:
29         check_palindrome()
```

Expected Output

Side-by-side comparison and observations

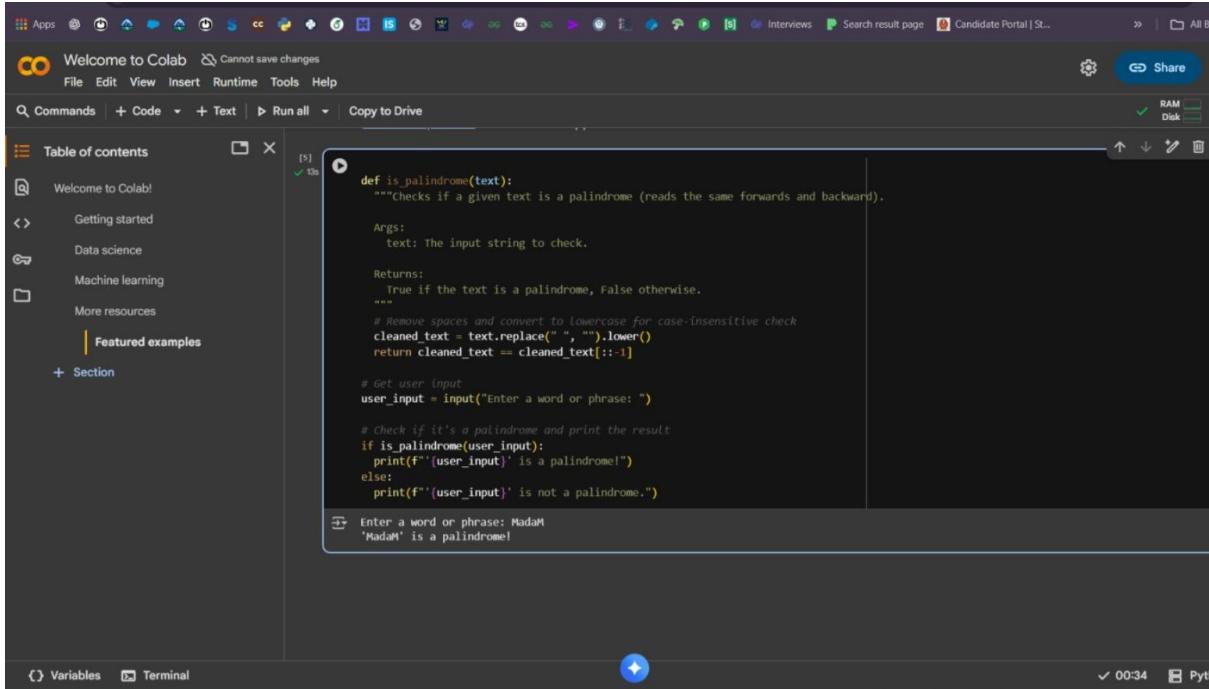
Practical output:



The screenshot shows the VS Code terminal window. It displays the command "python -u TASK 2.py" being run in the terminal, followed by the program's output. The output shows the user entering "MOM" and receiving the response "'MOM' is a palindrome!"

```
PS C:\Users\rimsha> python -u "c:\Users\rimsha\OneDrive\Desktop\python assessment\lab 2 assessment\task 2.py"
Enter text to check if it's a palindrome: MOM
"MOM" is a palindrome!
Enter text to check if it's a palindrome: 
```

Google colab:



The screenshot shows the Google Colab interface. On the left, there's a sidebar titled "Table of contents" with links to "Welcome to Colab!", "Getting started", "Data science", "Machine learning", "More resources", and "Featured examples". The main area contains a code cell with the following Python script:

```
def is_palindrome(text):
    """Checks if a given text is a palindrome (reads the same forwards and backward).

    Args:
        text: The input string to check.

    Returns:
        True if the text is a palindrome, False otherwise.
    """
    # Remove spaces and convert to lowercase for case-insensitive check
    cleaned_text = text.replace(" ", "").lower()
    return cleaned_text == cleaned_text[::-1]

# Get user input
user_input = input("Enter a word or phrase: ")

# Check if it's a palindrome and print the result
if is_palindrome(user_input):
    print(f"'{user_input}' is a palindrome!")
else:
    print(f"'{user_input}' is not a palindrome.")

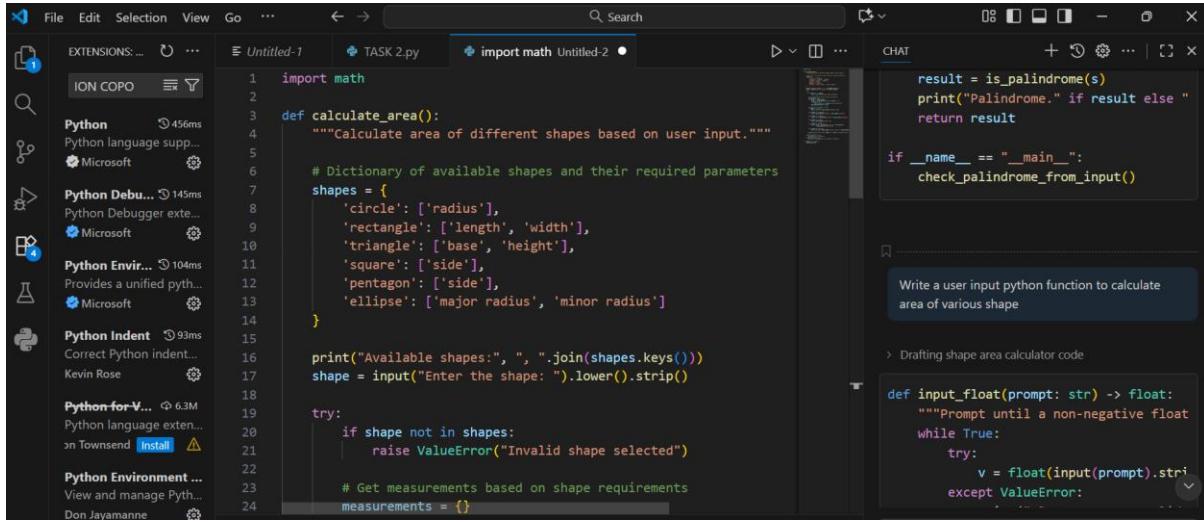

```

At the bottom of the code cell, there's an input placeholder: "Enter a word or phrase: MadaM". The status bar at the bottom right shows "00:34" and "Pyt".

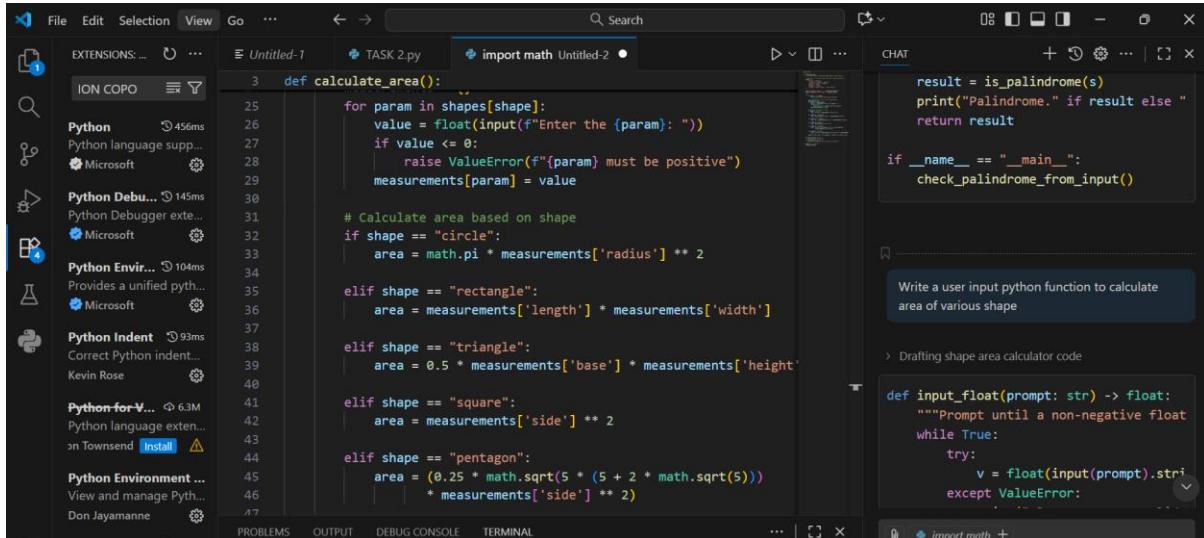
TASK DESCRIPTION -3

Detailed explanation with code snippet

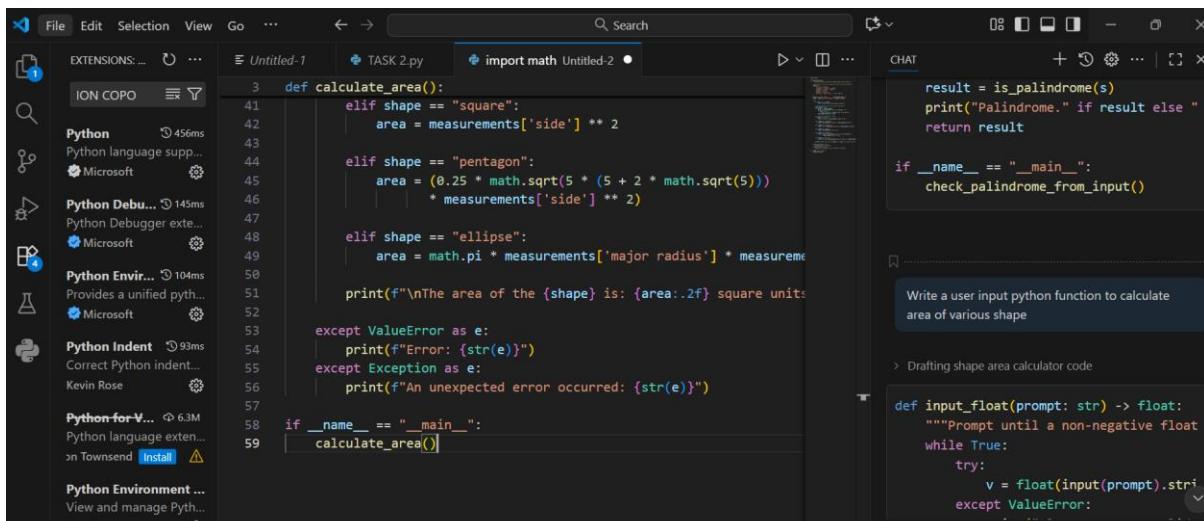
Prompt: Write a user input python function to calculate area of various shape



```
1 import math
2
3 def calculate_area():
4     """Calculate area of different shapes based on user input."""
5
6     # Dictionary of available shapes and their required parameters
7     shapes = {
8         'circle': ['radius'],
9         'rectangle': ['length', 'width'],
10        'triangle': ['base', 'height'],
11        'square': ['side'],
12        'pentagon': ['side'],
13        'ellipse': ['major radius', 'minor radius']
14    }
15
16    print("Available shapes: ", ", ".join(shapes.keys()))
17    shape = input("Enter the shape: ").lower().strip()
18
19    try:
20        if shape not in shapes:
21            raise ValueError("Invalid shape selected")
22
23        # Get measurements based on shape requirements
24        measurements = {}
```



```
3 def calculate_area():
4     for param in shapes[shape]:
5         value = float(input(f"Enter the {param}: "))
6         if value <= 0:
7             raise ValueError(f"{param} must be positive")
8         measurements[param] = value
9
10    # Calculate area based on shape
11    if shape == "circle":
12        area = math.pi * measurements['radius'] ** 2
13
14    elif shape == "rectangle":
15        area = measurements['length'] * measurements['width']
16
17    elif shape == "triangle":
18        area = 0.5 * measurements['base'] * measurements['height']
19
20    elif shape == "square":
21        area = measurements['side'] ** 2
22
23    elif shape == "pentagon":
24        area = (0.25 * math.sqrt(5 * (5 + 2 * math.sqrt(5))) *
25                * measurements['side'] ** 2)
```



```
3 def calculate_area():
4     elif shape == "square":
5         area = measurements['side'] ** 2
6
7     elif shape == "pentagon":
8         area = (0.25 * math.sqrt(5 * (5 + 2 * math.sqrt(5))) *
9                 * measurements['side'] ** 2)
10
11    elif shape == "ellipse":
12        area = math.pi * measurements['major radius'] * measurements['minor radius']
13
14    print(f"\nThe area of the {shape} is: {area:.2f} square units")
15
16    except ValueError as e:
17        print(f"Error: {str(e)}")
18    except Exception as e:
19        print(f"An unexpected error occurred: {str(e)}")
20
21    if __name__ == "__main__":
22        calculate_area()
```

Expected Output:

Detailed explanation with code snippet

Practical output:

```
∨ TERMINAL
python"
Available shapes: circle, rectangle, triangle, square, pentagon, ellipse
Available shapes: circle, rectangle, triangle, square, pentagon, ellipse
Enter the shape: CIRCLE
Enter the radius: 3
Enter the shape: CIRCLE
Enter the radius: 3

Enter the radius: 3

The area of the circle is: 28.27 square units
The area of the circle is: 28.27 square units
PS C:\Users\91910
```

TASK DESCRIPTION -4

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

Prompt: Generate a user input python function to calculate sum of square

The screenshot shows a code editor interface with several tabs open. The active tab is 'Untitled-2' containing Python code for calculating the sum of squares. The code includes documentation, a try-except block for user input validation, and a main loop for continuous calculations.

```
def sum_of_squares():
    """
    Calculate the sum of squares for a list of numbers provided by user.
    Returns the sum of squares and prints the calculation details.
    """
    try:
        # Get input from user
        numbers = input("Enter numbers separated by spaces: ").strip()

        # Convert strings to numbers and calculate squares
        numbers = [float(num) for num in numbers]
        squares = [num * num for num in numbers]

        # Calculate sum of squares
        total = sum(squares)

        # Print detailed output
        print("\nCalculations:")
        for i, num in enumerate(numbers):
            print(f"{num}² = {squares[i]}")
        print(f"\nSum of squares = {total}")

        return total
    except ValueError:
        print("Error: Please enter valid numbers separated by spaces")
```

```
for i, num in enumerate(numbers):
    print(f"{num}² = {squares[i]}")
print(f"\nSum of squares = {total}")

return total

except ValueError:
    print("Error: Please enter valid numbers separated by spaces")
    return None

if __name__ == "__main__":
    while True:
        sum_of_squares()

        # Ask if user wants to continue
        again = input("\nCalculate another sum of squares? (y/n): ")
        if again != 'y':
            break

    print("Thank you for using the sum of squares calculator!")
```

Expected Output

Screenshots of working environments with few prompts to generate python code

Practical output:

```
PS C:\Users\rimsha> python -u "C:\Users\rimsha\AppData\Local\Temp\tempCodeRunnerFile.py"
● Enter numbers separated by spaces: 3 4 5

Calculations:
3.02 = 9.0
4.02 = 16.0
5.02 = 25.0

Sum of squares = 50.0

Calculate another sum of squares? (y/n): 2
Thank you for using the sum of squares calculator!
○ PS C:\Users\rimsha>
```

TASK DESCRIPTION -5

Student need to write code to calculate sum of odd and even numbers in the list

Prompt: write code to calculate sum of odd and even numbers in the list

```
File Edit Selection View Go ... ← → Search
EXTENSIONS: ... C EXTENDED-1 TASK 2.py ● TASK 5.py X CHAT
C: > Users > rimsha > OneDrive > Desktop > python assessment > lab 2 assessment > TASK 5.py > ...
1 def calculate_odd_even_sums():
2     """
3         Calculate separate sums for odd and even numbers from user input.
4         Returns tuple of (odd_sum, even_sum).
5     """
6
7     try:
8         # Get input from user
9         numbers = input("Enter numbers separated by spaces: ").strip()
10        numbers = [int(num) for num in numbers]
11
12        # Calculate sums using list comprehension
13        even_sum = sum(num for num in numbers if num % 2 == 0)
14        odd_sum = sum(num for num in numbers if num % 2 != 0)
15
16        # Print results
17        print("\nResults:")
18        print(f"Even numbers sum: {even_sum}")
19        print(f"Odd numbers sum: {odd_sum}")
20
21        return (odd_sum, even_sum)
22
23    except ValueError:
24        print("Invalid input. Please enter valid integers separated by spaces.")
```

```
C:\> Users > rimsha > OneDrive > Desktop > python assessment > lab 2 assessment > TASK 5.py > ...
1 def calculate_odd_even_sums():
2     ...
3     print(f"Odd numbers sum: {odd_sum}")
4
5     return (odd_sum, even_sum)
6
7 except ValueError:
8     print("Error: Please enter valid integers separated by spaces")
9     return None
10
11 if __name__ == "__main__":
12     while True:
13         calculate_odd_even_sums()
14
15         # Ask if user wants to continue
16         again = input("\nCalculate another set of sums? (y/n): ").lower()
17         if again != 'y':
18             break
```

Expected Output: Refactored code written by student with improved logic
Practical Output:

```
python
Enter numbers separated by spaces: 4 7 2

Results:
Even numbers sum: 6
Odd numbers sum: 7

Results:
Even numbers sum: 6
Even numbers sum: 6
Odd numbers sum: 7
```