

ASSIGNMENT-2.5

Name-R.Teeneswari

Roll no – 2303A51932

Batch-30 Task-1:

Prompt: Write a program to calculate the sum of odd and even numbers in a list

Code:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "OPEN EDITORS" and "DEVOPS EXP 2,3,4".
- Code Editor:** Displays Python code named "ai 2.5.py".

```
1 #Write a Python program to calculate the sum of odd and even numbers in a list.
2 def sum_odd_even(numbers):
3     odd_sum = 0
4     even_sum = 0
5     for num in numbers:
6         if num % 2 == 0:
7             even_sum += num
8         else:
9             odd_sum += num
10    return odd_sum, even_sum
11
12 # Example usage
13 numbers = [1, 2, 3, 4, 5, 6]
14 odd_sum, even_sum = sum_odd_even(numbers)
15 print(f"Sum of odd numbers: {odd_sum}")
16 print(f"Sum of even numbers: {even_sum}")
```
- Terminal:** Shows the output of running the code:

```
[Running] python -u "c:/Users/heman/OneDrive/Desktop/DevOps/exp 2,3,4/ai 1932/ai 2.5 .py"
Sum of odd numbers: 9
Sum of even numbers: 12
```
- Status Bar:** Shows the file path as "master" and other status information.

Observation:

The **original code** works correctly but is written as a single block, making it harder to reuse and test.

The **refactored (AI-improved) code** separates logic into a function, improving:

- Readability
- Reusability
- Maintainability

Using a function allows the same logic to be reused with different lists without rewriting code.

Task-2:

Prompt: write a program explain a function that calculates the area of different shapes.

The code must include proper comments for explanation.

Code:

A screenshot of the Visual Studio Code interface. The left sidebar shows the file structure of a project named 'DEVOPS EXP 2,3,4'. The main editor window contains Python code for calculating the area of shapes. The code uses a function named `calculate_area` which takes a shape name and dimensions as parameters. It includes comments explaining the logic for each shape type: circle, rectangle, and triangle. The output tab at the bottom shows the execution of the script and its results.

```
File Edit Selection View Go Run Terminal Help ← → Q DevOps exp 2,3,4
OPEN EDITORS JS index.js 1, U AI ASS Coding U Untitled-1 AI-ASS Coding.ipynb U ai 2.5.py X ai 2.5.py U
EXPLORER ai 1932 > ai 2.5.py > ...
index.js 1, U
AI ASS Coding U
Untitled-1
AI-ASS Coding... U
ai 2.5.py ai 1932 U
ai 2.5.py ai 1932 U
DEVOPS EXP 2,3,4
.github\workflow...
ciyaml
> zencoder
zenflow\workflows
ai 1932
ai 2.5.py U
ai 2.5.py U
node_modules
.gitignore
AI ASS Coding U
AI-ASS Coding.ipc... U
Dockerfile
index.js 1, U
package-lock.json U
package.json M
PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE PORTS SPELL CHECKER
[Running] python -u "c:\Users\heman\OneDrive\Desktop\DevOps exp 2,3,4\ai 1932\ai 2.5 .py"
File "c:\Users\heman\OneDrive\Desktop\DevOps exp 2,3,4\ai 1932\ai 2.5 .py", line 63
IndentationError: unindent does not match any outer indentation level
[Done] exited with code=1 in 0.291 seconds
[Running] python -u "c:\Users\heman\OneDrive\Desktop\DevOps exp 2,3,4\ai 1932\ai 2.5 .py"
Sum of odd numbers: 9
Sum of even numbers: 12
Sum of even numbers: 24
Sum of odd numbers: 9
OUTLINE TIMELINE
y5 master* 0 2 0 Live Share
Ln 76, Col 1 Spaces:4 UTF-8 CR/LF {} Python 3.13.7 ⓘ Go Live Zencoder Monica

```

CODE

Observation:

This program uses **one function** to calculate the area of **multiple shapes**, which avoids code duplication.

The shape parameter decides **which formula** to apply.

The function uses **conditional statements** (if / elif) to select the correct formula.

It improves **code clarity**, making onboarding easier and faster.

Task:3

Prompt: explain a function that calculates the area of different shapes (cursor used)

Shapes. Write a program to find the sum of even and odd numbers in a list Code:

```

File Edit Selection View Go Run Terminal Help < > Q DevOps exp 2,3,4
OPEN EDITORS JS index.js 1, U AI ASS Coding U Untitled-1 AI-ASS Coding.ipynb U ai 2.5.py X ai 2.2.py U
EXPLORER ai 1932 > ai 2.5.py > ...
ai 1932
57 def main():
58     #write a program to find the sum of even and odd numbers in a list
59     numbers = list(map(int, input("Enter numbers separated by space: ").split()))
60     even_sum=0
61     odd_sum = 0
62     for num in numbers:
63         if num % 2 == 0:
64             even_sum += num
65         else:
66             odd_sum += num
67     print(f"Sum of even numbers: {even_sum}")
68     print(f"Sum of odd numbers: {odd_sum}")
69
70
71
72
73
74
75
76 if __name__ == "__main__":
77
78
79
80
81

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS SPELL CHECKER Filter (e.g. text, excludeText, t...)

[Done] exited with code=1 in 0.291 seconds

[Running] python -u "c:\users\heman\OneDrive\Desktop\DevOps exp 2,3,4\ai 1932\ai 2.5 .py"

Sum of odd numbers: 9
Sum of even numbers: 12
Sum of even numbers: 12
Sum of odd numbers: 9

[Done] exited with code=0 in 0.271 seconds

In 76 Col 1 Spaces: 4 UTF-8 CRLF [] Python Go Live Zencoder ✓ Monica

Observation:

The program demonstrates **how one function can handle multiple use cases**.

Comments clearly explain:

What the function does

Why each condition exists

What each parameter represents

Using comments makes the code **junior-developer friendly**, which is ideal for onboarding.

The main () function separates **user interaction** from **business logic**, improving structure.

This style is considered **clean, readable, and professional** in real-world projects. Task-4:

Prompt: Based on practical usage and experimentation, compare **Gemini**, **GitHub Copilot**, and **Cursor AI** in terms of **usability** and **code quality**.

Observation:

Gemini is best suited for **explanations and learning support**. It produces readable, beginner-friendly code and clear step-by-step reasoning, making it ideal for onboarding juniors and understanding concepts.

GitHub Copilot excels in **real-time coding assistance** inside IDEs. It is fast, context-aware, and highly productive for experienced developers, but its code may lack explanations.

Cursor AI stands out for **prompt sensitivity and refactoring quality**. It responds strongly to detailed prompts, generating cleaner, more structured, and optimized code, making it suitable for improving legacy codebases.

usability, Copilot integrates seamlessly into workflows, Gemini is conversational and educational, and Cursor AI offers powerful prompt-driven refactoring.

code quality, Cursor AI and Copilot generally produce more professional, production-ready code, while Gemini focuses on clarity over optimization