TECHNIK NEST

INNOVATIVE MINDS, NESTING SUCCESS

Name: Musfira Ahmed

Intern ID: TN/IN01/PY/002

Email ID : ahmedmusfira3@gmail.com

Task week: 06

Internship Domain: Python Development

Instructor Name: Mr. Hassan Ali

Task 01

Use math & statistics libraries to get square roots and average.

What I Did (Step by Step)

- Imported the math library to calculate square roots.
- Imported the statistics library to calculate the average of a list.
- Used user-defined data to demonstrate both functions.

Code Screenshot

```python
import math
import statistics

Numbers = [4, 9, 16, 25, 36]

SquareRoots = [math.sqrt(num) for num in Numbers]
print("Square Roots:", SquareRoots)

Average = statistics.mean(Numbers)
print("Average of the list:", Average)
```


Output Screenshot

```
PS D:\Python Internship Tasks\HB_Tasks\Week 06 tasks with report> & "C:/Program Files/Python313/python3.13t.exe" "d:/Python Internship Tasks/HB_Tasks/Week 06 tasks with report/Math_Statistics_Operations (Square root_Average).py"
Square Roots: [2.0, 3.0, 4.0, 5.0, 6.0]
Average of the list: 18
PS D:\Python Internship Tasks\HB_Tasks\Week 06 tasks with report>
```

Learning and Challenges

- Learned how to use built-in Python libraries like math and statistics.

- Faced Challenge when applying square root to a list solved it using a list comprehension **([math.sqrt(num) for num in list]).**
- Understood how to calculate statistical data from a list of numbers

---

Task 02

Create a custom package and import it in another script.

What I Did (Step by Step)

1. Create the Package Folder

Create a folder named mypackage.

2. Create __init__.py

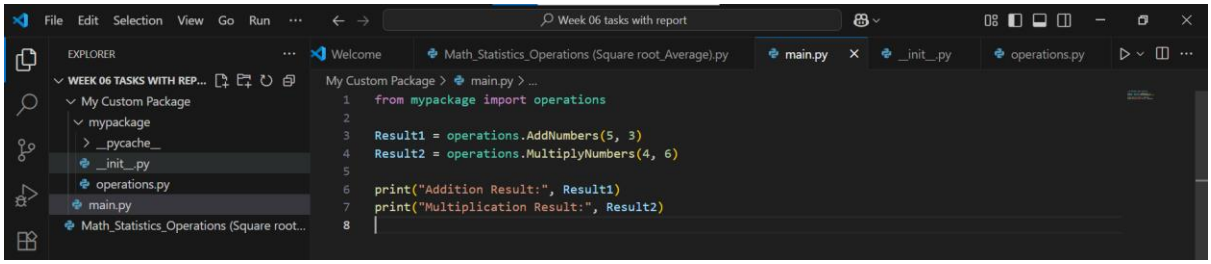
Inside mypackage/, create an empty **__init__.py** file (this marks it as a Python package).

3. Create operations.py

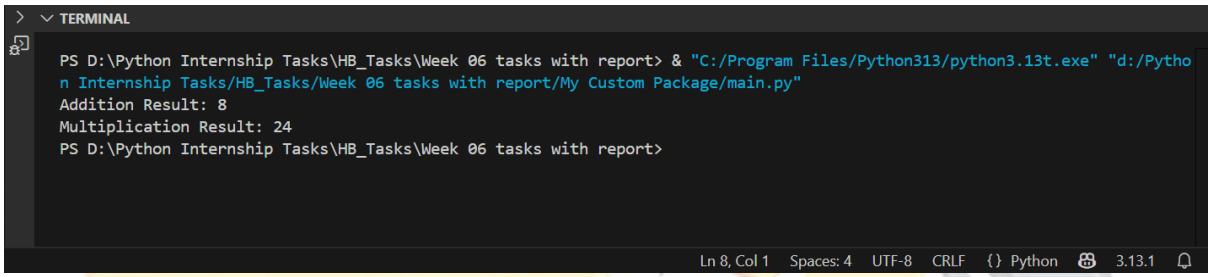
This will contain your custom functions.

mypackage/operations.py:

Code Screenshot



Output Screenshot



Learning and Challenges

- Learned how to modularize Python code using packages.
- Understood how **__init__.py** is required to recognize folders as packages.
- Faced issue with import paths solved by ensuring folder structure is correct and script is run from the root directory.

Task 03

Create a virtual environment, install requests & numpy, and print their versions.

What I Did (Step by Step)

Step 1: Create a Virtual Environment

Open terminal or command prompt:

python -m venv myenv

This creates a folder myenv/ containing the virtual environment.

Step 2: Activate the Virtual Environment

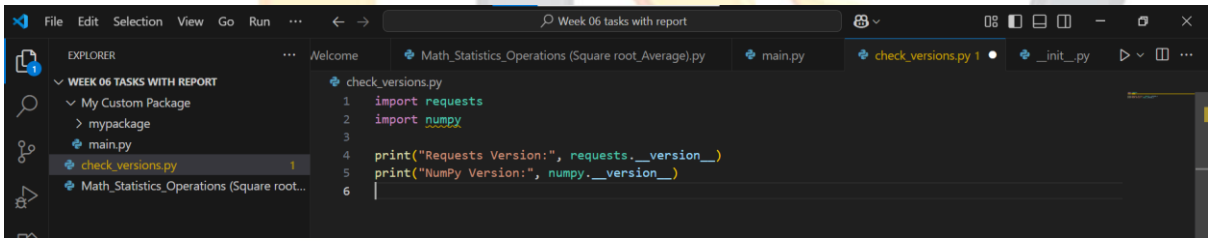
- On Windows:

myenv\Scripts\activate

Step 3: Install Packages

pip install requests numpy

Code Screenshot



Output Screenshot

```
Select Administrator: Command Prompt

(myenv) C:\Windows\system32>pip install requests numpy
Collecting requests
  Using cached requests-2.32.4-py3-none-any.whl.metadata (4.9 kB)
Collecting numpy
  Downloading numpy-2.3.2-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting charset_normalizer<4,>=2 (from requests)
  Using cached charset_normalizer-3.4.2-cp313-cp313-win_amd64.whl.metadata (36 kB)
Collecting idna<4,>=2.5 (from requests)
  Using cached idna-3.10-py3-none-any.whl.metadata (10 kB)
Collecting urllib3<3,>=1.21.1 (from requests)
  Using cached urllib3-2.5.0-py3-none-any.whl.metadata (6.5 kB)
Collecting certifi>=2017.4.17 (from requests)
  Using cached certifi-2025.7.14-py3-none-any.whl.metadata (2.4 kB)
Using cached requests-2.32.4-py3-none-any.whl (64 kB)
Downloading numpy-2.3.2-cp313-cp313-win_amd64.whl (12.8 MB)
   ---------------------------------------- 12.8/12.8 MB 432.8 kB/s eta 0:00:00
Using cached certifi-2025.7.14-py3-none-any.whl (162 kB)
Using cached charset_normalizer-3.4.2-cp313-cp313-win_amd64.whl (105 kB)
Using cached idna-3.10-py3-none-any.whl (70 kB)
Using cached urllib3-2.5.0-py3-none-any.whl (129 kB)
Installing collected packages: urllib3, numpy, idna, charset_normalizer, certifi, requests
Successfully installed certifi-2025.7.14 charset_normalizer-3.4.2 idna-3.10 numpy-2.3.2 requests-2.32.4 urllib3-2.5.0

[notice] A new release of pip is available: 24.3.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(myenv) C:\Windows\system32>
```

```
Administrator: Command Prompt

(myenv) D:\Python Internship Tasks\HB_Tasks\Week 06 tasks with report>python check_versions.py
Requests version: 2.32.4
NumPy version: 2.3.2

(myenv) D:\Python Internship Tasks\HB_Tasks\Week 06 tasks with report>
(myenv) D:\Python Internship Tasks\HB_Tasks\Week 06 tasks with report>
```

Learning and Challenges

- Learned how to isolate Python environments using **venv**.
- Practiced package installation with pip.
- Discovered how to check package versions using **__version__**.
- Faced issue on Windows with activation solved by running terminal as administrator.

Task 04

Print list of all installed pip packages from Python code.

What I Did (Step by Step)

Step 1: Activate Virtual Environment

If your virtual environment is already created (e.g., myenv), activate it:

myenv\Scripts\activate

Prompt changes to:

(myenv) C:\>

Step 2: Create Python Script

Create a new file named **list_installed.py** and paste the code above into it. Save it inside your working folder.

Step 3: Install Required Module

If running the script gives the error:

ModuleNotFoundError: No module named 'pkg_resources'

Then install setuptools inside the virtual environment:

pip install setuptools

This provides access to the pkg_resources module.

Step 4: Run the Script

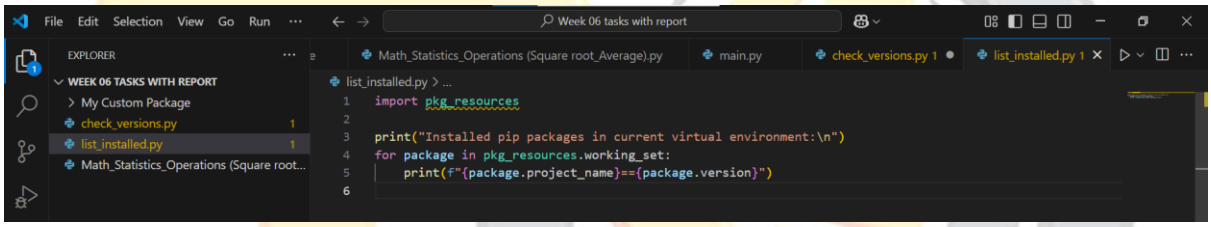
Navigate to the script's location:

cd "D:\Python Internship Tasks\HB_Tasks\Week 06 tasks with report"

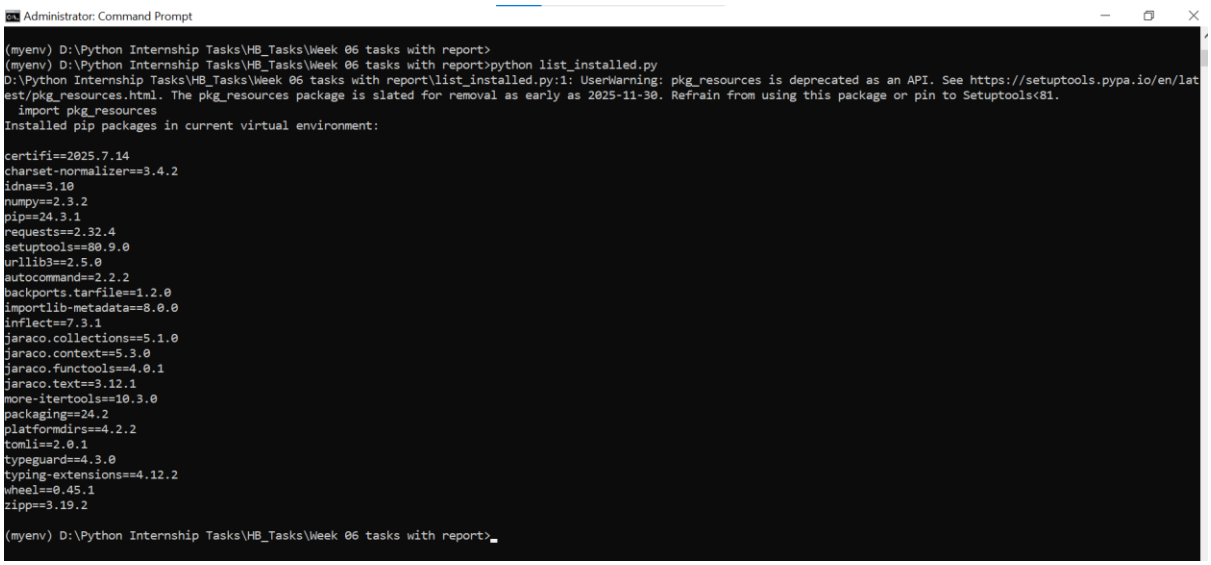
Then run:

python list_installed.py

Code Screenshot



Output Screenshot



Learning and Challenges

- How to inspect installed **pip packages** inside a virtual environment using Python code.
- **pkg_resources** module was not found
- Ran the script from the wrong directory
- Forgot to activate the virtual environment before running the script
- pkg_resources provides access to all installed packages.
- Importance of **setuptools** in Python environments.
- Managing and troubleshooting Python virtual environments.
- Navigating between folders and using Python from the command line.

Task 05

Create Gradio app that takes a number and returns its square.

What I Did (Step by Step)

1. Install Gradio (if not installed)

pip install gradio

2. Save the code in a file

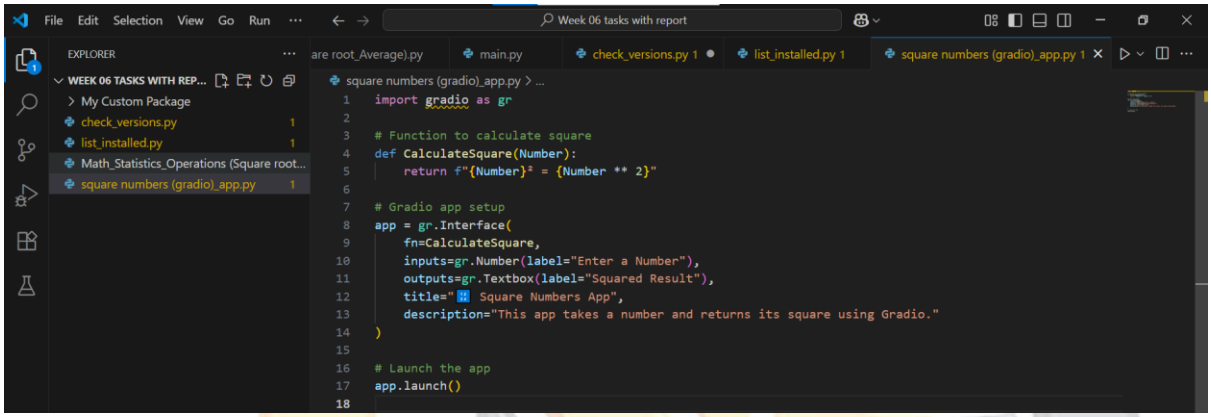
   Name the file square_numbers_app.py

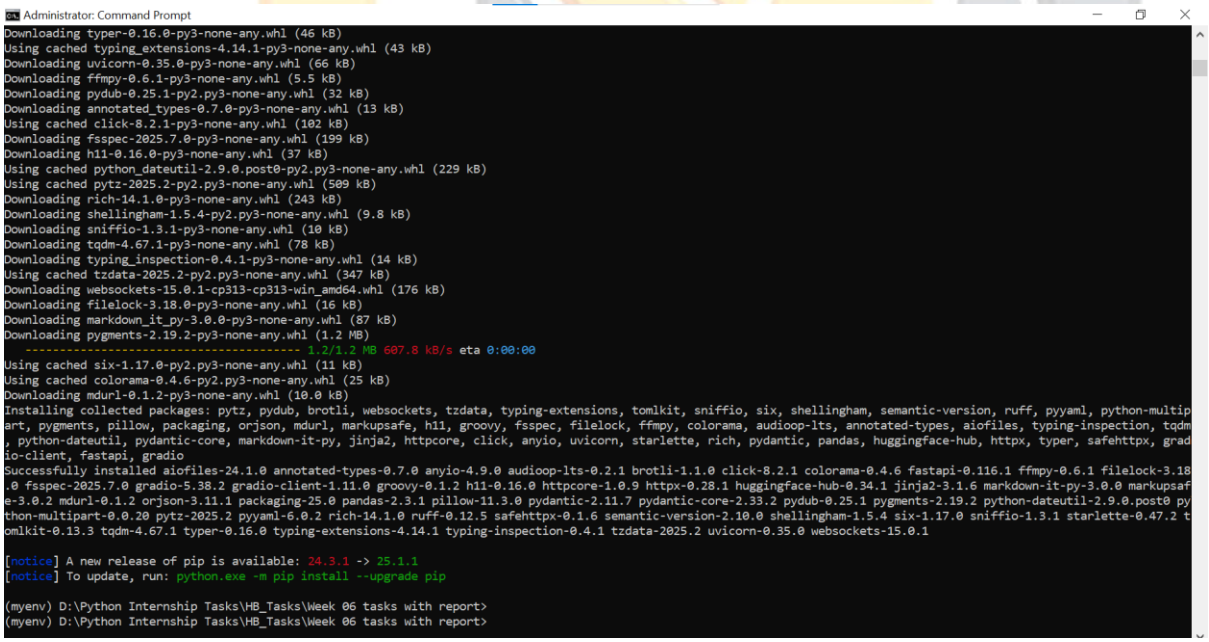
3. Run the file in terminal

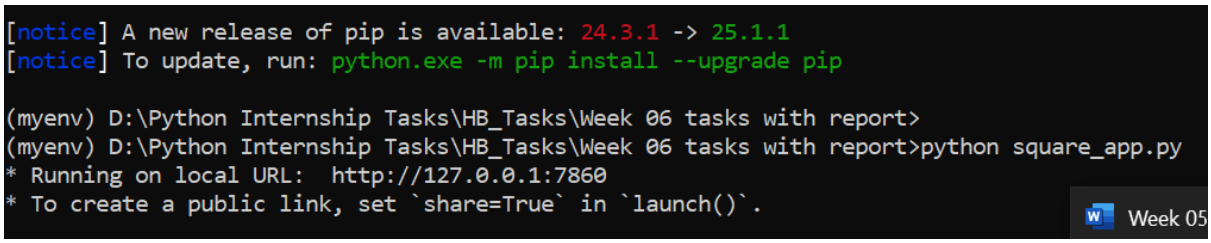
python square_numbers_app.py

4. A web browser will open with a simple UI.

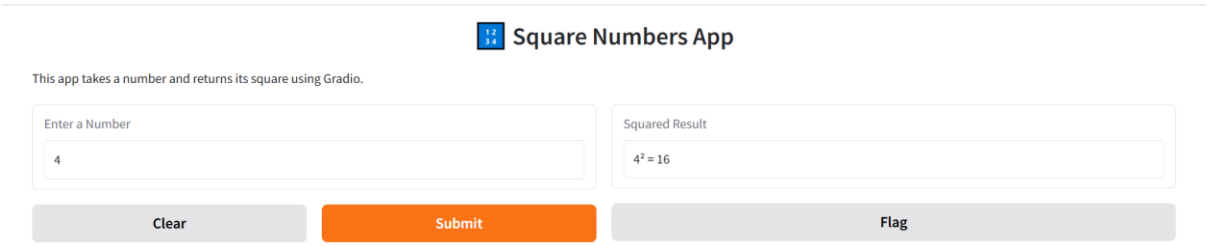
Code Screenshot







Output Screenshot



Learning and Challenges

- Understood how to use the Gradio library to build a quick web-based interface.
- Learned how to define a Python function that takes a number input and returns its square.
- Implemented **gr.Interface()** to connect the function with the user interface.
- First time using Gradio
- Entered string instead of a number (caused error)
- App didn't auto-open in browser

Task 06

Create Gradio interface that takes a sentence and returns it reversed.

What I Did (Step by Step)

How to Run:

1. Save the code in a file called **reverse_sentence_app.py.**
2. Open your terminal or VS Code terminal.
3. Navigate to the folder where your file is saved.
4. Activate your virtual environment:

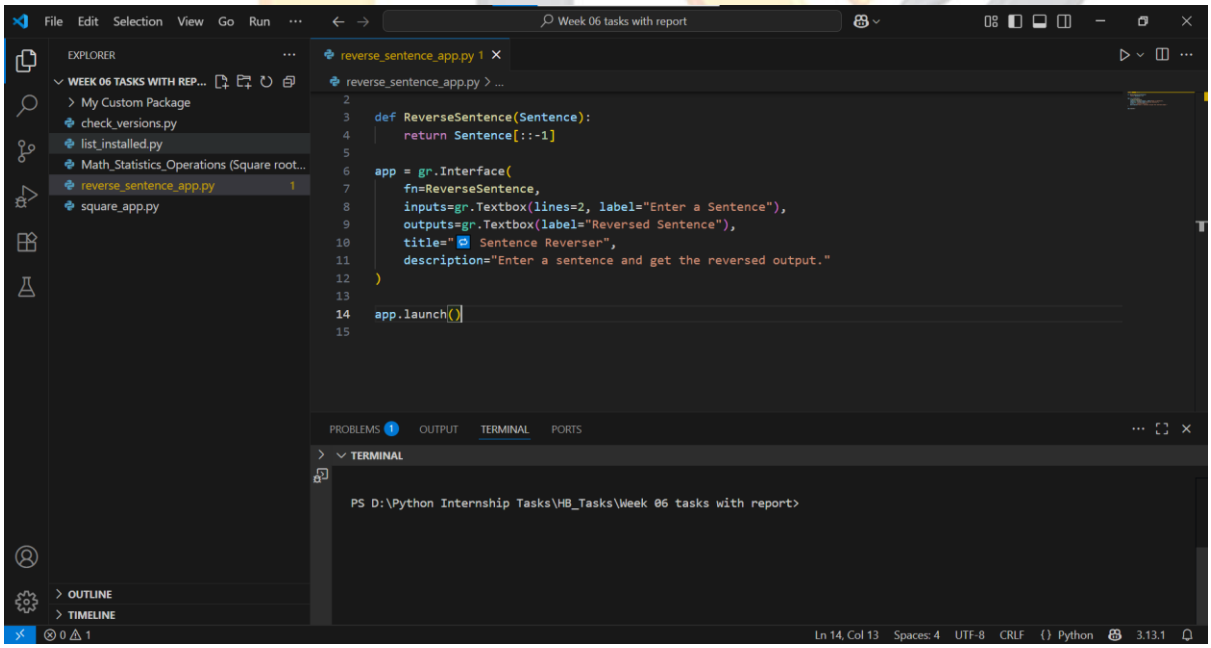
myenv\Scripts\activate # For Windows

5. Run the script:

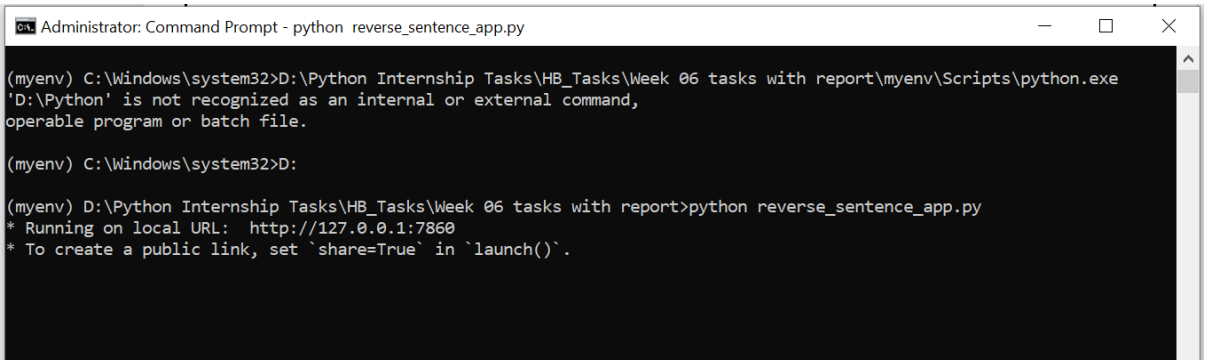
python reverse_sentence_app.py

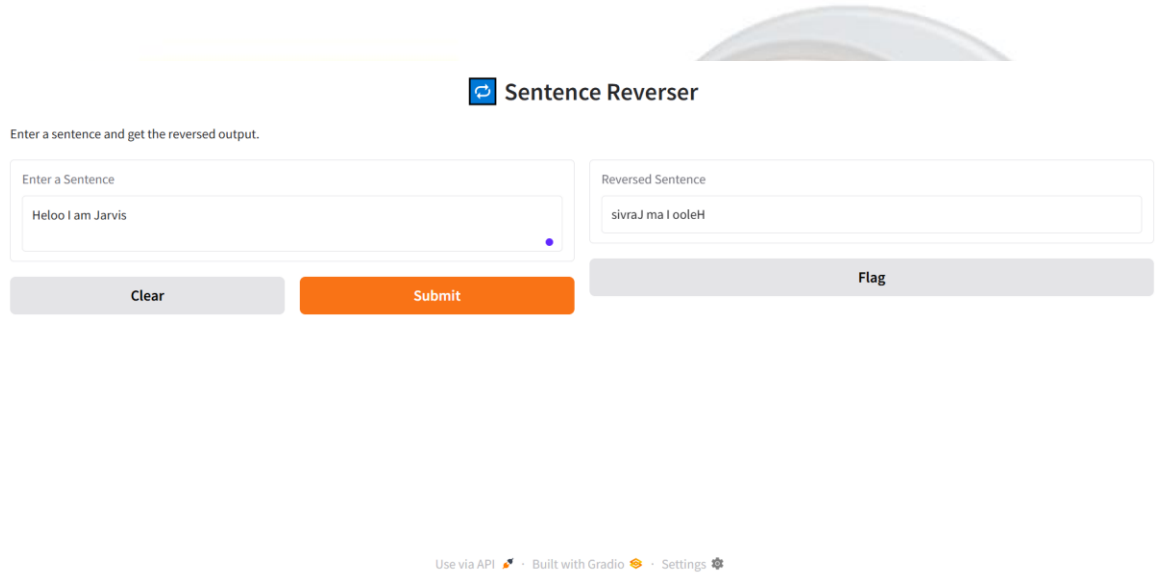
6. Open the browser and go to**: http://127.0.0.1:7860**

Code Screenshot



Output Screenshot





Learning and Challenges

- Learned how to use Gradio to quickly build a web interface for a Python function.
- Practiced using string slicing **([::-1])** to reverse a sentence.
- Understood how Gradio maps function inputs and outputs to a web interface.

- Learned how to run Gradio apps locally in a virtual environment.
- Gradio module not found.
- Forgetting to activate virtual environment.
- App not loading at **127.0.0.1.**