**Meta Scifor Technologies**

**Streamlit Theory Test**

1. **What is Streamlit and what are its main features?**

**=>** Streamlit is an open-source Python library that makes it easy to create and share custom web applications for machine learning and data science projects.

**Here are the main features of Streamlit** :-

 **Simplicity and Ease of Use**: Streamlit allows you to create web apps with minimal code.

 **Interactive Widgets**: Streamlit offers a variety of widgets (such as sliders, buttons, and text inputs) that enable users to interact with the app and dynamically update visualizations and data.

 **Real-Time Updates**: Apps built with Streamlit update in real-time as users interact with them, providing an interactive and engaging user experience.

 **Data Display**: It provides easy ways to display data using tables, charts, and graphs.

 **Media Integration**: Streamlit supports the inclusion of images, audio, and video, allowing for rich media content within your applications.

 **Easy Deployment**: Streamlit apps can be deployed easily using Streamlit's sharing platform or other cloud services.

 **Custom Themes**: Streamlit allows customization of app themes to better match your branding or personal preference.

**2) How does Streamlit differ from other web application frameworks like Flask or Django?**

**=>**  **Streamlit** is best for rapid prototyping and creating interactive data science applications with minimal code.

 **Flask** is a flexible, lightweight framework suitable for a wide range of web applications, offering more control but requiring more setup for interactivity and front-end development.

 **Django** is a full-featured framework designed for building complex, scalable applications, providing many built-in tools and features but with a steeper learning curve and more setup requirements.

**3) What are some typical use cases for Streamlit?**

**=>**  **Data Exploration and Analysis**:

* Quickly create interactive dashboards to explore datasets.
* Visualize data using built-in charting capabilities or integrate with libraries like Matplotlib, Seaborn, and Plotly.
* Allow users to filter and manipulate data through interactive widgets.

 **Machine Learning Model Deployment**:

* Deploy machine learning models with interactive interfaces for inputting data and viewing predictions.
* Facilitate parameter tuning with sliders and other controls.

 **Real-Time Data Monitoring**:

* Develop dashboards to monitor real-time data streams, such as stock prices, sensor data, or web traffic.
* Update visualizations in real-time based on incoming data.

 **Reporting and Presentation**:

* Generate dynamic reports that can be shared as interactive web applications.
* Combine text, media, and interactive visualizations to present findings in a compelling manner.

**4) How do you create a simple Streamlit app?**

**=>** Below is a step-by-step guide to creating a basic Streamlit app :-

**Step 1: Install Streamlit**

First, you need to install Streamlit. You can do this using pip

### Step 2: Create a Python Script

Create a new Python script, for example, app.py. This script will contain the code for your Streamlit app.

### Step 3: Import Streamlit

In your app.py file, start by importing the Streamlit library

### Step 4: Add Content to the App

You can add various elements to your Streamlit app, such as text, sliders, and charts.

### Step 5: Run the Streamlit App

To run your Streamlit app, open a terminal, navigate to the directory containing your app.py file, and execute the following command:

streamlit run app.py

This command will start a local web server and open your Streamlit app in a new browser tab.

### Step 6: Interact with Your App

Now you can interact with the app in your web browser. You’ll see the title, text, slider, DataFrame, and line chart you added.

**5) Can you explain the basic structure of a Streamlit script?**

### => Basic Structure of a Streamlit Script

1. **Import Libraries**:
   * Start by importing Streamlit and any other necessary libraries (e.g., Pandas, NumPy).
2. **Title and Header**:
   * Use st.title() and st.header() to set the main title and section headers of your app.
3. **Text and Markdown**:
   * Use st.write() or st.markdown() to display text, explanations, and descriptions.
4. **Interactive Widgets**:
   * Add interactive elements like sliders, buttons, and input boxes using Streamlit's built-in widgets (st.slider(), st.button(), st.text\_input(), etc.).
5. **Display Data**:
   * Show data using tables and charts. Streamlit integrates seamlessly with libraries like Pandas for dataframes (st.dataframe(), st.table()) and visualization libraries like Matplotlib, Seaborn, and Plotly (st.line\_chart(), st.bar\_chart(), etc.).
6. **Control Flow**:
   * Use control flow statements (e.g., if, for, while) to make your app dynamic and responsive to user inputs.
7. **Media**:
   * Include images, audio, and video using st.image(), st.audio(), and st.video().
8. **Execution and Display**:
   * Ensure that the script is executed by Streamlit using the command streamlit run your\_script.py.

**6) How do you add widgets like sliders, buttons, and text inputs to a Streamlit app?**

**=> Adding Sliders:**

import streamlit as st

slider\_value = st.slider("Select a value", 0, 100, 50)

st.write("Selected value:", slider\_value)

**Adding Buttons:**

import streamlit as st

if st.button("Click me"):

st.write("Button clicked!")

**Adding Text Inputs:**

import streamlit as st

user\_input = st.text\_input("Enter some text")

st.write("You entered:", user\_input)

**Select Box:**

import streamlit as st

option = st.selectbox( 'Select an option', ('Option 1', 'Option 2', 'Option 3') )

st.write('You selected:', option)

**Checkbox:**

import streamlit as st

checkbox = st.checkbox("Check me")

if checkbox:

st.write("Checkbox is checked")

else:

st.write("Checkbox is not checked")

**Radio Buttons**:

import streamlit as st

radio\_option = st.radio( 'Choose one:', ('Option A', 'Option B', 'Option C'))

st.write('You chose:', radio\_option)

**7) How does Streamlit handle user interaction and state management?**

**=>**  **User Interaction**:

* Streamlit provides interactive widgets (e.g., sliders, buttons, text inputs) to capture user input.
* Each interaction with a widget causes the script to re-run from top to bottom.

 **State Management**:

* By default, Streamlit scripts are stateless; variables do not persist across interactions.
* Use st.session\_state to persist variables and manage state across interactions.
* Use st.cache\_data to cache expensive computations and improve performance.

**8) What are some best practices for organizing and structuring a Streamlit project?**

### => 1. Project Directory Structure

A well-organized directory structure makes your project more manageable and readable.

### 2. Use Separate Modules for Different Components

Break down your application into separate modules to improve readability and reusability. For example, keep utility functions in one module, sidebar components in another, and page components in their own modules.

### 3. Use st.session\_state for State Management

Manage application state using st.session\_state to keep track of variables across user interactions.

### 4. Cache Expensive Computations

Use st.cache\_data to cache expensive computations, such as data loading or complex calculations, to improve performance.

### 5. Use Functions for Reusability

Encapsulate repeated or complex code in functions to improve reusability and readability.

### 7. Documentation and Comments

Document your code and write meaningful comments to explain the functionality. Maintain a README.md file to provide an overview of the project, setup instructions, and usage guidelines.

**9) How would you deploy a Streamlit app locally?**

**=>** Step-by-Step Guide to Deploy Locally

**1. Set Up Your Project Environment:**

- Ensure you have Python installed. You can download Python from [python.org](https://www.python.org/).

- Set up a virtual environment to manage dependencies.

```bash

python -m venv env

source env/bin/activate # On Windows: env\Scripts\activate

```

**2. Install Streamlit:**

- Install Streamlit and any other dependencies your app requires using pip.

```bash

pip install streamlit

```

- If you have a `requirements.txt` file, you can install all dependencies listed in it:

```bash

pip install -r requirements.txt

```

**3. Create Your Streamlit App:**

- Write your Streamlit app code in a Python file, typically named `app.py`.

```python

import streamlit as st

st.title("My Streamlit App")

st.write("Hello, world!")

```

**4. Run the Streamlit App:**

- Use the Streamlit CLI to run your app locally. Open your terminal or command prompt, navigate to the directory containing your `app.py` file, and run:

```bash

streamlit run app.py

```

**5. Access the App in Your Browser:**

- After running the above command, Streamlit will start a local web server and open the app in your default web browser. By default, the app will be accessible at `http://localhost:8501`.

**10) Can you describe the steps to deploy a Streamlit app?**

=> **1. Set Up Your Project Environment:**

- Ensure you have Python installed. You can download Python from [python.org](https://www.python.org/).

- Set up a virtual environment to manage dependencies.

```bash

python -m venv env

source env/bin/activate # On Windows: env\Scripts\activate

```

**2. Install Streamlit:**

- Install Streamlit and any other dependencies your app requires using pip.

```bash

pip install streamlit

```

- If you have a `requirements.txt` file, you can install all dependencies listed in it:

```bash

pip install -r requirements.txt

```

**3. Create Your Streamlit App:**

- Write your Streamlit app code in a Python file, typically named `app.py`.

```python

import streamlit as st

st.title("My Streamlit App")

st.write("Hello, world!")

```

**4. Run the Streamlit App:**

- Use the Streamlit CLI to run your app locally. Open your terminal or command prompt, navigate to the directory containing your `app.py` file, and run:

```bash

streamlit run app.py

```

**5. Access the App in Your Browser:**

- After running the above command, Streamlit will start a local web server and open the app in your default web browser. By default, the app will be accessible at `http://localhost:8501`.

**11) What is the purpose of the requirements.txt file in the context of Streamlit deployment?**

**=> Purpose of requirements.txt**

1. **Dependency Management**: The requirements.txt file lists all Python packages that your Streamlit app depends on. This includes Streamlit itself and any other libraries (like pandas, numpy, matplotlib, etc.) that your app uses for data processing, visualization, or other functionalities.
2. **Environment Reproducibility**: By listing specific versions of each package, requirements.txt ensures that anyone deploying your Streamlit app, whether locally or on a cloud platform, can create a consistent environment with the same dependencies. This helps in avoiding compatibility issues and ensures that your app behaves predictably across different environments.
3. **Deployment Automation**: Many deployment platforms, such as Heroku, AWS Elastic Beanstalk, and Google Cloud Run, use requirements.txt to automatically install dependencies when deploying your Streamlit app. This simplifies the deployment process and ensures that all required packages are installed correctly.