



Published in CodeX



Afaque Umer

Follow

Sep 25, 2022 · 5 min read · Listen



Save



Streamlit🔥+ FastAPI⚡ - The ingredients you need for your next Data Science Recipe

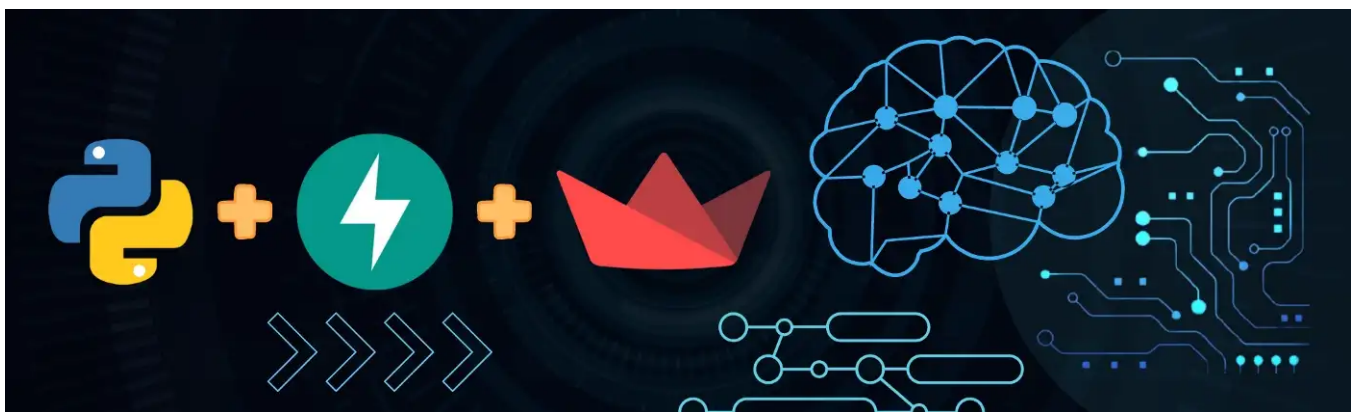


Image by Author

Streamlit is an open-source, free, all-python framework to rapidly build and share interactive dashboards and web apps for Data Science / Machine Learning Projects.

Streamlit's slogan is *"The fastest way to build and share data apps"*, which is true. This framework makes it easy for Data Scientists and Machine Learning engineers to create powerful user interfaces and **convert the jupyter notebook/ python scripts into interactive apps** and **need not worry about views, templates, and then JS** to make it interactive, you simply write a python script and it runs from **top to bottom**.

Although Streamlit can be used in production, it's best for rapid prototyping. By serving up the Machine learning / Data Science model with **FastAPI**, you can quickly move to a production-ready UI with Dash or Bokeh after the prototype is approved.



844



11



In this short blog post I'll cover the following steps:

1. Create a basic Python Calculator module.
2. Serve the function from the module using FastAPI.
3. Creating a very basic UI using Streamlit.
4. Integrating Streamlit & FastAPI.

I'm not going for any fancy machine learning model here, once we are good with the know-how of FastAPI + Streamlit pipeline, we can **scale it up with any data science/machine learning models**.

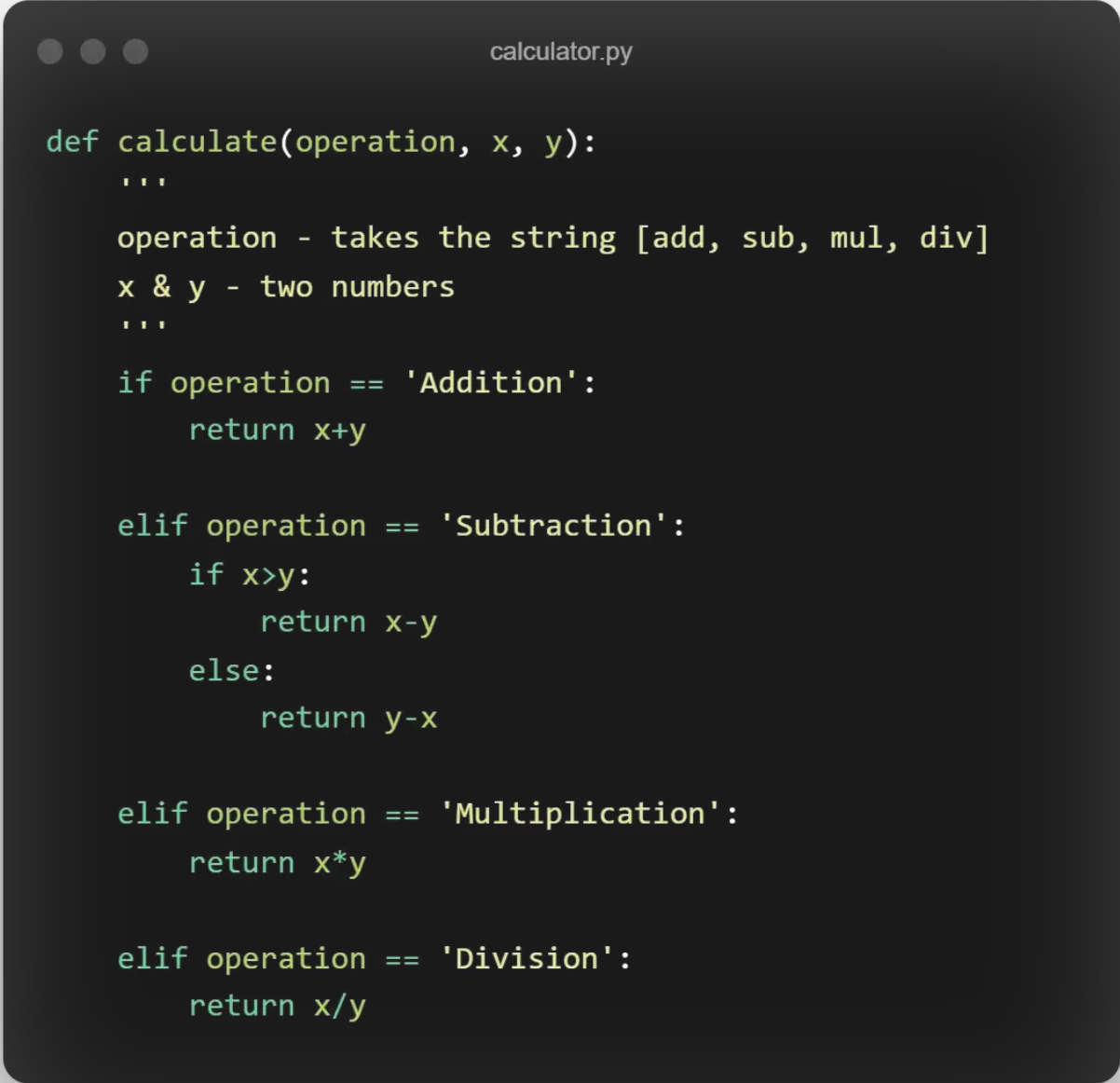


Step 1: Create the calculator module

A module is just a python script containing a set of functions you want to include in your application. Here we are building a calculator app so the inputs we need are the numbers and the operation we want to perform.

Let's create a function *calculate* for serving the same purpose with parameters *operation*, *x* & *y* respectively and save this as *calculator.py* module.

It's a very basic function that simply performs the operation against the operation name passed using the *if-elif* blocks and returns the result.



```
def calculate(operation, x, y):
    """
    operation - takes the string [add, sub, mul, div]
    x & y - two numbers
    """
    if operation == 'Addition':
        return x+y

    elif operation == 'Subtraction':
        if x>y:
            return x-y
        else:
            return y-x

    elif operation == 'Multiplication':
        return x*y

    elif operation == 'Division':
        return x/y
```

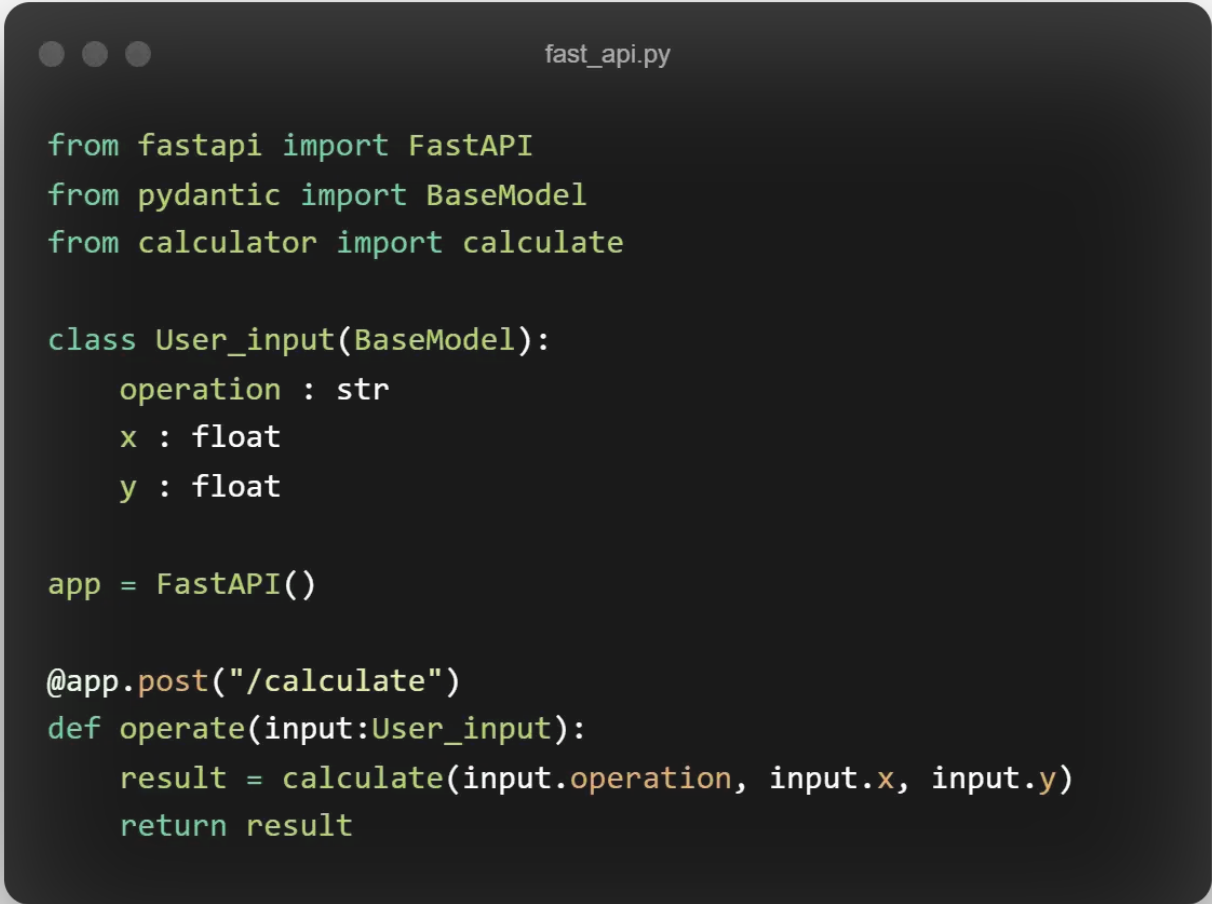
calculator module with calculate function: Image by Author

Step 2: Creating the API using FastAPI ⚡

Want to brush up on FastAPI fundamentals? Read this [!\[\]\(cbe2492b119e39e02a1dab2af4a4b296_img.jpg\)](#)

Here we'll create a basic request body. We'll create a class *User_input* using pydantic's BaseModel for type validation and sending the arguments to the

calculator function, which we will simply import from the *calculator module* we created above.



```
fast_api.py

from fastapi import FastAPI
from pydantic import BaseModel
from calculator import calculate

class User_input(BaseModel):
    operation : str
    x : float
    y : float

app = FastAPI()

@app.post("/calculate")
def operate(input:User_input):
    result = calculate(input.operation, input.x, input.y)
    return result
```

FastAPI script: Image by Author

For running the FastAPI server we need to run the following command:

```
uvicorn fast_api:app --reload
```

We can now open the Swagger UI here 🖱️ <http://127.0.0.1:8000/docs>

default

POST /calculate Operate

Parameters

No parameters

Request body *required* application/json

```
{
  "operation": "string",
  "x": 0,
  "y": 0
}
```

Execute

Swagger UI: Image by Author

Let's test our path operation in Thunder Client.

POST http://127.0.0.1:8000/calculate Send

Status: 200 OK Size: 4 Bytes Time: 9 ms

Query Headers ² Auth Body ¹ Tests

Json Xml Text Form Form-encode GraphQL Binary

Json Content Format

```
1 {
2   "operation": "Addition",
3   "x": 4.4,
4   "y": 5.6
5 }
```

Response Headers ⁵ Cookies Results

1 10.0

Request & response: Image by Author



This completes our back-end process. We have created a module and created an API, invoking function by importing the module.

One of the main benefits of REST APIs is that the **REST protocol separates the data storage(back-end) and the UI(front-end) from the server** thus allowing the client and server to be independent.

Step 3. Create the web app using Streamlit 🔥

For creating the UI we'll use Streamlit, it's an application framework that makes it easy for data scientists and machine learning engineers to create data scripts into shareable web apps in minutes. All in pure Python. No front-end experience is required ✨.



You can install streamlit via pip:

pip install streamlit

You can learn more about Streamlit from here 🖱️ <https://streamlit.io/>

```
stream_lit.py

import streamlit as st
import json
import requests

st.title("Basic Caculator App 🧮")

# taking user inpputs
option = st.selectbox('What operation You want to perform?',
                      ('Addition', 'Subtraction', 'Multiplication', 'Division'))

st.write("")
st.write("Select the numbers from slider below 🗲️")
x = st.slider("X", 0, 100, 20)
y = st.slider("Y", 0, 130, 10)

#converting the inputs into a json format
inputs = {"operation": option, "x": x, "y": y}

# when the user clicks on button it will fetch the API
if st.button('Calculate'):
    res = requests.post(url = "http://127.0.0.1:8000/calculate", data = json.dumps(inputs))

    st.subheader(f"Response from API 🚀 = {res.text}")
```


Streamlit app script: Image by Author

In the above app I have used the following components:

st.title: Display text in title formatting.

st.write: Write arguments to the app.

st.selectbox: Display a select widget.

st.slider: Display a slider widget.

st.button: Display a button widget.

For running the streamlit server we need to run the following command:

```
streamlit run stream_lit.py
```

It will launch the app in the browser, you can go to <http://localhost:8501> The web app will look something like this 📌

Basic Caculator App

What operation You want to perform?

Addition

Select the numbers from slider below 📌

X



Y



Calculate

Streamlit App: Image by Author

So, our UI is created in just a few lines of code. This will allow the user to select an operation from the select box, using sliders the value of x & y will be selected (by default the values are defined as 20 & 10 respectively). It should show some results after pressing the calculate button.

Step 4: Assemble Frontend with Backend



For avoiding errors & running the app well we need to make sure that both our front-end & back-end servers are up & running.

```
# For running streamlit app
streamlit run stream_lit.py

# For running backend with FastAPI
uvicorn fast_api:app --reload
```

st.button is a boolean component, it returns a **True** if the button was clicked on the last run of the app, and **False** otherwise. We will utilize this inside the **if-else** block & we will send a request only when the button will be clicked. This will avoid unnecessary operations in larger apps and can save a lot of computational loss in machine learning apps.

We will use Python's request library for integrating the backend in the streamlit script. In the streamlit app, we are storing the inputs into a dictionary variable

```
inputs = {"operation": option, "x": x, "y": y}
```

JSON is a syntax for storing and exchanging data. If you have a Python object, you can convert it into a JSON by using the `json.dumps()` method.

We will convert our python object *inputs* into a JSON format while sending requests to our API.

```
requests.post(url = "http://127.0.0.1:8000/calculate",
              data = json.dumps(inputs))
```

We have already tested the response to our request in STEP 2. So we will simply write the response in our app as a result of the operation.

Basic Caculator App

What operation You want to perform?

Multiplication

Select the numbers from slider below 🗨️

X



Y



Calculate

Response from API 🚀 = 2000.0



Congratulations !!! Now we know how to create a fully functional web app.

I will try to bring up more **Machine learning/Data science concepts** and will try to break down fancy-sounding terms and concepts into simpler ones.

I hope you enjoyed this article! You can **follow me** Afaque Umer for **more** such articles.

Thanks for reading 🙏

Keep learning 🧠 Keep Sharing 🤝 Stay Awesome 🤘

Python

Machine Learning

Data Science

Artificial Intelligence

Deployment

Sign up for CrunchX

By CodeX

A weekly newsletter on what's going on around the tech and programming space [Take a look.](#)

Emails will be sent to shivakmuddam25@gmail.com. [Not you?](#)

