Introduction to Python & Flask

Explanation of the Flask framework and how a Flask app works

We'll cover the following

The Flask framework

If we don't understand what we are dealing with, it will be difficult to troubleshoot the problems we have going forward.

Let's understand some terminologies used in Python:

Modules

In short, imports in Python allow us to use pre-coded classes and functions with just one line of the statement. The ***import*** keyword is used to access code written in modules. eg. from <modulename> import <class>

In our code, the first line in app.py is:

```
from flask import Flask, render_template, request, redirect
```

This imports the Flask class from the Flask module and different other functionalities.

Routes

Frameworks encapsulate the response mechanism in routes. To access a particular resource, we need a particular URI/address. Routes provide the facility to deliver functionality at a particular endpoint without creating separate files with the extensions .php, .py or .html.

In our code, we have:

```
@app.route('/', methods=['POST', 'GET'])
```

which creates index routes that can be accessed by visiting a site directly eg. https://<website name>/ which is a GET request. It can also post data using a POST request.

Python package index

All the prebuilt Python modules are present in this repository. We have to install the module to use in our code.

The requirements.txt file in our code lists all the modules we will need to run and these modules can be installed using the `pip install -r requirements.txt` command in the terminal.

Virtual environments

Since Python is open source and used in system programming, different versions of Python can be installed on the same system. Virtual environments isolate our Python environment from system libraries so that we can safely play around with our project.

Using natively-installed Python for developing projects can sometimes leave inconsistencies in your system libraries.

We will use a virtual environment for running the app to show how these were used before the advent of Docker.

I think this will knowledge will help you understand the code better. Feel free to learn more about Flask on the web.

Now, let's see how to run the app.

Running the app on Windows

If you are using a Windows machine, make sure you have Python 3 installed.

If you don't, please follow this tutorial to install it.

Once you have it installed, add a Python 3 path to your environment variables so that you can access Python from anywhere.

Steps to run

- Open a command prompt/PowerShell window
- type `python3 -m venv myenvname`
- for cmd, type `<venv>\Scripts\activate.bat`
- for PowerShell, type `<venv>\Scripts\Activate.ps1`
- `git clone https://github.com/venky8283/Docker.git`
- `cd Docker`
- `pip install -r requirements.txt`
- `set FLASK_APP=app.py` set this to let Flask know where the Flask instance is located
- `flask run`

Once you are done, you should be able to access the app on 'http://localhost:5000'.

Running the app on a Mac/Linux machine

Most of the time, Python 3 comes prebuilt with these platforms. If yours isn't, install Python 3 using this link.

Steps to run

The steps here are similar to what we used with a Windows machine, except the command to activate your virtual environment.

- Open a terminal
- type `python3 -m venv myenvname`
- type `source <venv>\bin\activate`
- `git clone https://github.com/venky8283/Docker.git`
- `cd Docker`
- `pip install -r requirements.txt`
- `export FLASK_APP=app.py` set this to let Flask know where the Flask instance is located (optional: if you have a 'flaskrun' file with this variable)

located (optional. If you have a ".flaskrun" file with this variable).

- `flask run`

Once you are done, you should be able to access the app on 'http://localhost:5000'.

> Running so many commands every time you set a project is cumbersome, plus these steps will increase once the project gets bigger. Also, remembering different commands for different platforms is difficult. That's where Docker comes to the rescue.

In the next lesson, we will build our first Dockerfile.