30 Days Of Python: Day 26 - Python for web

Python is a general purpose programming language and it can be used for many places. In this section, we will see how we

use Python for the web. There are many Python web frame works. Django and Flask are the most popular ones. Today, we will see how to use Flask for web development.

Flask

Flask is a web development framework written in Python. Flask uses Jinja2 template engine. Flask can be also used with other modern front libraries such as React.

If you did not install the virtualenv package yet install it first. Virtual environment will allows to isolate project dependencies from the local machine dependencies.

Folder structure

After completing all the step, your project file structure should look like this:

```
├── Procfile
├── app.py
├── env
│   ├── bin
├── requirements.txt
├── static
│   └── css
│       └── main.css
```

```
└── templates
    ├── about.html
    ├── home.html
    ├── layout.html
    ├── post.html
    └── result.html
```

Setting up your project directory

Follow the following steps to get started with Flask.

Step 1: install virtualenv using the following command.

```
pip install virtualenv
```

Step 2:

```
/Desktop$ mkdir python_for_web
Desktop$ cd python_for_web/
Desktop/python_for_web$ virtualenv venv
Desktop/python_for_web$ source venv/bin/activate
(env) Desktop/python_for_web$ pip freeze
(env) Desktop/python_for_web$ pip install Flask
(env) Desktop/python_for_web$ pip freeze
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
(env) Desktop/python_for_web$
```

We created a project director named python_for_web. Inside the project we created a virtual environment *venv* which could be any name but I prefer to call it *venv*. Then we activated the virtual environment. We used pip freeze to check the installed packages in the project directory. The result of pip freeze was empty because a package was not installed yet.

Now, let's create app.py file in the project directory and write the following code. The app.py file will be the main file in the project. The following code has flask module, os module.

Creating routes

The home route.

```
# let's import the flask
from flask import Flask
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    return '<h1>Welcome</h1>'
```

```
@app.route('/about')
def about():
    return '<h1>About us</h1>'


if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

To run the flask application, write python app.py in the main flask application directory.

After you run *python app.py* check local host 5000.

Let us add additional route. Creating about route

```
# let's import the flask
from flask import Flask
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    return '<h1>Welcome</h1>'

@app.route('/about')
def about():
    return '<h1>About us</h1>'

if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Now, we added the about route in the above code. How about if we want to render an HTML file instead of string? It is possible to render HTML file using the function *render_templae*. Let us create a folder called templates and create home.html and about.html in the project directory. Let us also import the *render_template* function from flask.

Creating templates

Create the HTML files inside templates folder.

home.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
```

```
    <title>Home</title>
  </head>

  <body>
    <h1>Welcome Home</h1>
  </body>
</html>
```

about.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>About</title>
  </head>

  <body>
    <h1>About Us</h1>
  </body>
</html>
```

Python Script

app.py

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    return render_template('home.html')

@app.route('/about')
def about():
    return render_template('about.html')

if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

As you can see to go to different pages or to navigate we need a navigation. Let's add a link to each page or let's create a layout which we use to every page.

Navigation

```
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/about">About</a></li>
</ul>
```

Now, we can navigate between the pages using the above link. Let us create additional page which handle form data. You can call it any name, I like to call it post.html.

We can inject data to the HTML files using Jinja2 template engine.

```
# let's import the flask
from flask import Flask, render_template, request, redirect, url_for
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    techs = ['HTML', 'CSS', 'Flask', 'Python']
    name = '30 Days Of Python Programming'
    return render_template('home.html', techs=techs, name = name, title =
'Home')

@app.route('/about')
def about():
    name = '30 Days Of Python Programming'
    return render_template('about.html', name = name, title = 'About Us')

@app.route('/post')
def post():
    name = 'Text Analyzer'
    return render_template('post.html', name = name, title = name)


if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Let's see the templates too:

home.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Home</title>
  </head>

  <body>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
    </ul>
    <h1>Welcome to {{name}}</h1>
     <ul>
    {% for tech in techs %}
      <li>{{tech}}</li>
    {% endfor %}
```

```
    </ul>
  </body>
</html>
```

about.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>About Us</title>
  </head>

  <body>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
    </ul>
    <h1>About Us</h1>
    <h2>{{name}}</h2>
  </body>
</html>
```

Creating a layout

In the template files, there are lots of repeated codes, we can write a layout and we can remove the repetition. Let's create layout.html inside the templates folder. After we create the layout we will import to every file.

Serving Static File

Create a static folder in your project directory. Inside the static folder create CSS or styles folder and create a CSS stylesheet. We use the *url_for* module to serve the static file.

layout.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <link

href="https://fonts.googleapis.com/css?family=Lato:300,400|Nunito:300,400|R
aleway:300,400,500&display=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='css/main.css') }}"
    />
    {% if title %}
```

```
    <title>30 Days of Python - {{ title}}</title>
    {% else %}
    <title>30 Days of Python</title>
    {% endif %}
  </head>

  <body>
    <header>
      <div class="menu-container">
        <div>
          <a class="brand-name nav-link" href="/">30DaysOfPython</a>
        </div>
        <ul class="nav-lists">
          <li class="nav-list">
            <a class="nav-link active" href="{{ url_for('home')
}}">Home</a>
          </li>
          <li class="nav-list">
            <a class="nav-link active" href="{{ url_for('about')
}}">About</a>
          </li>
          <li class="nav-list">
            <a class="nav-link active" href="{{ url_for('post') }}"
              >Text Analyzer</a
            >
          </li>
        </ul>
      </div>
    </header>
    <main>
      {% block content %} {% endblock %}
    </main>
  </body>
</html>
```

Now, lets remove all the repeated code in the other template files and import the layout.html. The href is using *url_for* function with the name of the route function to connect each navigation route.

home.html

```
{% extends 'layout.html' %} {% block content %}
<div class="container">
  <h1>Welcome to {{name}}</h1>
  <p>
    This application clean texts and analyse the number of word, characters
and
    most frequent words in the text. Check it out by click text analyzer at
the
    menu. You need the following technologies to build this web
application:
  </p>
  <ul class="tech-lists">
    {% for tech in techs %}
    <li class="tech">{{tech}}</li>

    {% endfor %}
  </ul>
</div>
```

```
{% endblock %}
```

about.html

```
{% extends 'layout.html' %} {% block content %}
<div class="container">
  <h1>About {{name}}</h1>
  <p>
    This is a 30 days of python programming challenge. If you have been
coding
    this far, you are awesome. Congratulations for the job well done!
  </p>
</div>
{% endblock %}
```

post.html

```
{% extends 'layout.html' %} {% block content %}
<div class="container">
  <h1>Text Analyzer</h1>
  <form action="https://thirtydaysofpython-v1.herokuapp.com/post"
method="POST">
    <div>
      <textarea rows="25" name="content" autofocus></textarea>
    </div>
    <div>
      <input type="submit" class="btn" value="Process Text" />
    </div>
  </form>
</div>

{% endblock %}
```

Request methods, there are different request methods(GET, POST, PUT, DELETE) are the common request methods which allow us to do CRUD(Create, Read, Update, Delete) operation.

In the post, route we will use GET and POST method alternative depending on the type of request, check how it looks in the code below. The request method is a function to handle request methods and also to access form data. app.py

```
# let's import the flask
from flask import Flask, render_template, request, redirect, url_for
import os # importing operating system module

app = Flask(__name__)
# to stop caching static file
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0



@app.route('/') # this decorator create the home route
def home ():
    techs = ['HTML', 'CSS', 'Flask', 'Python']
    name = '30 Days Of Python Programming'
    return render_template('home.html', techs=techs, name = name, title =
'Home')

@app.route('/about')
def about():
```

```
    name = '30 Days Of Python Programming'
    return render_template('about.html', name = name, title = 'About Us')

@app.route('/result')
def result():
    return render_template('result.html')

@app.route('/post', methods= ['GET','POST'])
def post():
    name = 'Text Analyzer'
    if request.method == 'GET':
         return render_template('post.html', name = name, title = name)
    if request.method =='POST':
        content = request.form['content']
        print(content)
        return redirect(url_for('result'))

if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

So far, we have seen how to use template and how to inject data to template, how to a common layout. Now, lets handle static file. Create a folder called static in the project director and create a folder called css. Inside css folder create main.css. Your main. css file will be linked to the layout.html.

You don't have to write the css file, copy and use it. Let's move on to deployment.

Deployment

Creating Heroku account

Heroku provides a free deployment service for both front end and fullstack applications. Create an account on heroku and install the heroku CLI for you machine. After installing heroku write the following command

Login to Heroku

```
 heroku login
heroku: Press any key to open up the browser to login or q to exit:
```

Let's see the result by clicking any key from the keyboard. When you press any key from you keyboard it will open the heroku login page and click the login page. Then you will local machine will be connected to the remote heroku server. If you are connected to remote server, you will see this.

```
$ heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/browser/be12987c-583a-
4458-a2c2-ba2ce7f41610
Logging in... done
Logged in as Suniksha@gmail.com
```

```
$
```

Create requirements and Procfile

Before we push our code to remote server, we need requirements

- requirements.txt
- Procfile

```
(env) /Desktop/python_for_web$ pip freeze
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
(env) Suniksha@Suniksha:~/Desktop/python_for_web$ touch requirements.txt
(env) Suniksha@Suniksha:~/Desktop/python_for_web$ pip freeze >
requirements.txt
(env) Suniksha@Suniksha:~/Desktop/python_for_web$ cat requirements.txt


Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
(env) Suniksha@Suniksha:~/Desktop/python_for_web$ touch Procfile
(env) Suniksha@Suniksha:~/Desktop/python_for_web$ ls
Procfile          env/              static/
app.py            requirements.txt  templates/
(env) Suniksha@Suniksha:~/Desktop/python_for_web$
```

The Procfile will have the command which run the application in the web server in our case on Heroku.

```
web: python app.py
```

Pushing project to heroku

Now, it is ready to be deployed. Steps to deploy the application on heroku

1. git init
2. git add .
3. git commit -m "commit message"
4. heroku create 'name of the app as one word'
5. git push heroku master
6. heroku open(to launch the deployed application)

After this step you will get an application like this

Exercises: Day 26

1. You will build this application. Only the text analyser part is left

🎉 CONGRATULATIONS ! 🎉