Python Flask Tutorial- 1

- Given an empty python file in our project directory we create a basic flask instance. With the front page of flask documentation....we copy and paste into file.

- __name__ is a special variable in python that is just the name of the module. That is so flask knows where to look for templates, static files etc.

- Now we have an instantiated flask application, and we can create our routes .
  Routes are what we type into browsers to go to different pages. We create these using route decorators.
- Decorators are ways to add further functionality to existing functions.

- In this case the app.route decorator will handle all of the backend stuff and simply allow us to write a function that returns the information that will be shown on our website for this specific route.

- When we start our application and navigate to our home page it will display Hello World.

- We start with the command line, navigate to project directory with cd Flask_Blog/
  This puts us in the same directory where the FlaskBlog.py module lives. Before we run our application we need to set
  an environment variable to the file that we want to be our flask application.
  This is done through:
  export FLASK_APP = flaskblog.py

- With the environment variable set we can run the flask application by saying - flask run
  If this worked successfully there should be feedback below the shows the IP address of local machine where flask is running. 5000 is the port number.
  In order to reach the running flask application, copy and paste the http address.

```
6
7   from flask import Flask
8   app = Flask(__name__)
9
10  @app.route("/")
11  def hello():
12      return "Hello World!"
13
14
15
16
    Run Cell | Run Above | Debug Cell
17  # %%
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Sams-Air:Flask_Blog sammarcustaylor$ /usr/local/bin/python3 /Users/sammarcustaylor/Desktop/Flask_Blog/Flask_Blog.py
Sams-Air:Flask_Blog sammarcustaylor$ /usr/local/bin/python3 /Users/sammarcustaylor/Desktop/Flask_Blog/Flask_Blog.py
Sams-Air:Flask_Blog sammarcustaylor$ /usr/local/bin/python3 /Users/sammarcustaylor/Desktop/Flask_Blog/flaskblog.py
Sams-Air:Flask_Blog sammarcustaylor$ /usr/local/bin/python3 /Users/sammarcustaylor/Desktop/Flask_Blog/flaskblog.py
Sams-Air:Flask_Blog sammarcustaylor$ pwd
/Users/sammarcustaylor/Desktop/Flask_Blog
Sams-Air:Flask_Blog sammarcustaylor$ ls
Flask_Blog.ipynb        flaskblog.py
Sams-Air:Flask_Blog sammarcustaylor$ export FLASK_APP=flaskblog.py
Sams-Air:Flask_Blog sammarcustaylor$ flask run
 * Serving Flask app "flaskblog.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- There is also an alias for the ip address known as local host, so if the 127….
  Gets replaced with http://localhost:5000/ it works the same.

- When we change the Hello World to have header tags and then check back we can see that the changes did not take effect so we need to stop the web server and then run it again.
- This can be done through
    1. Ctrl + c
    2. flask run
- Keep in mind in Chrome and other web browsers you can view page source code through opposite clicking on the page and view source code.
- When working on the blog you will be making a lot of changes and it would be a pain to restart the web server each time therefore we can have the server show changes without restarting our app by running in debug mode

```
Sams-Air:Flask_Blog sammarcustaylor$ export FLASK_DEBUG=1
Sams-Air:Flask_Blog sammarcustaylor$ flask run
 * Serving Flask app "flaskblog.py" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 304-804-143
```

- We see the additional information that was not there before. When you refresh the local host page in the browser then we can see the app is working and reflecting live changes. Given that we are in debug mode.
- If you do not want to work with environment variables you run the app directly using Python stating a conditional

```
4   # In[1]:
5
6
7   from flask import Flask
8   app = Flask(__name__)
9
10  @app.route("/")
11  def hello():
12      return "<h1>Home Page!</h1>"
13
14  if __name__ == "__main__":
15      app.run(debug = True)
16
```

- If we run this script with Python directly the name will be main, but if we import this module to somewhere else then the name will be the name of our moule. Therefore this conditional will only be true if we run the script directly.
- This allows us to simply run the python script, and we will get a similar output.

- If we tried to navigate to an about page without any route it will return not found. So we must input a route within the code.

```
4   # In[1]:
5
6
7   from flask import Flask
8   app = Flask(__name__)
9
10  @app.route("/")
11  def hello():
12      return "<h1>Home Page!</h1>"
13
14  @app.route("/about")
15  def about():
16      return "<h1>About Page!</h1>"
17
18  if __name__ == "__main__":
19      app.run(debug = True)
20
```

- 404 not found error is nullified and the page has a route.

- If we wanted to have multiple routes handled by the same function then it is as simple as having another decorator.
- The code below has two http routes that return to the same page. Each will return home page in a header tag.

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
@app.route("/home")
def home():
    return "<h1>Home Page!</h1>"

@app.route("/about")
def about():
    return "<h1>About Page!</h1>"

if __name__ == "__main__":
    app.run(debug = True)
```

PART 2

- Instead of writing the HTML code directly into the routes we will utilize templates. We refer to the templates within the routes
- We create a file in the same directory as “templates' ' then we create two empty html files home and about. Within the home file we put the Home Page title in the body.
- Then we render the template in the return portion of the route. So we must import render_template. Then place in the return.

```html
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <h1>Home Page</h1>
</body>
</html>
```

```python
from flask import Flask, render_template
app = Flask(__name__)

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

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

if __name__ == "__main__":
    app.run(debug = True)
```

- View page source to confirm changes to html structure. The same is one with the about route page.
- Let's say that we had posts that we wanted to display in our template….So we create a variable called posts. Posts contains a list of dictionaries.
- We can pass in an argument into the render_template function to have access to that variable in our template. So now we switch over to the home template in order to see how to loop through that data.

```html
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    {% for post in posts %}
        <h1>{{ post.title}}</h1>
        <p>By {{post.author}} on {{post.date_posted}}</p>
        <p>{{post.content}}</p>
    {% endfor %}
</body>
</html>
```

- Within home.html we open up a code block in the initial for loop. We get the posts variable by passing in posts = posts in our render_template statement.

Blog Post 1

By Sam Taylor on January 8th, 2021

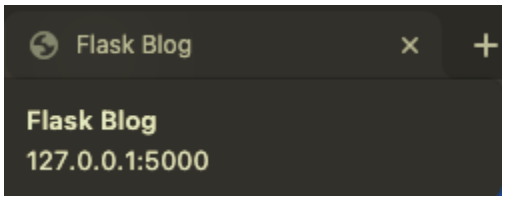
First Post Content

Blog Post 2

By Jane Doe on January 9th, 2021

Second Post Content

- The result is shown above as our data that was specified within the html was looped over. View the page source to actually see the html.
- We can also do if, else conditionals as well. So say we wanted to pass a title to our webpage, but if we didn't pass a title then we wanted to just use a default of flask blog.
- We pass a title into about template, but not home. Therefore, the home template should go through the if statement and see it does not have a title, then it defaults to Flask Blog.
- In the about template it will see a title and then use that title instead.



- There is one thing to point out that is not very good design. ABout and home page templates have a lot of repeated code. This is not good because if we want to make a change in one place then we have to make that change at all occurrences within the code. It is better to have everything repeated in a single place.

- For example if we wanted to change the default title we would have to change in both templates. So the solution is to create a new html file and pick out all pieces that are in common.
- We solve this through template inheritance. We create a new file called layout.html
  Here we pick out the repeated section between home and about template. The only things that are different are the body sections. We delete the part that is unique to the about page which is the entire body tag. So now we only have the html that is shared between both of our routes.
- Now we must create a block, and that is a section that the child templates can override. Therefore we place the block in the layout.html body section. This allows us to override the content section in our other templates but inherit what we have in common.
- In the home.html we only keep the for loop that is unique to that page, and then we place a code block with the extends keyword in order to extend the layout.html content
  Then we want the for loop content to override the content block of that layout template. To do that we can wrap in a content block by saying block content, and endblock content.

templates > layout.html > html

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       {% if title %}
5           <title>Flask Blog - {{ title }}</title>
6       {% else %}
7           <title>Flask Blog</title>
8       {% endif %}
9   </head>
10  <body>
11      {% block content %}{% endblock %}
12  </body>
13  </html>
```

templates > home.html > ...

```
{% extends "layout.html" %}
{% block content %}
    {% for post in posts %}
        <h1>{{ post.title}}</h1>
        <p>By {{post.author}} on {{post.date_posted}}</p>
        <p>{{post.content}}</p>
    {% endfor %}
{% endblock content %}
```

templates > about.html > ...

```
{% extends "layout.html" %}
{% block content %}
    <h1>About Page Homie!</h1>
{% endblock content %}
```

- Now we will implement bootstrap into the main layout.html in order to provide some nice framework. We do this by navigating to https://getbootstrap.com/docs/5.1/getting-started/introduction/
  And then copying and pasting into the head, and body tag. Also we have created a div class container in the body while the blocks are still valid.

templates > layout.html > html > body > div.container

```
<!DOCTYPE html>
<html>
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" inte


    {% if title %}
        <title>Flask Blog - {{ title }}</title>
    {% else %}
        <title>Flask Blog</title>
    {% endif %}
</head>
```

```
</head>
<body>
    <div class="container">
        {% block content %}{% endblock %}
    </div>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha3

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js" integrity="sha384-
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js" integrity="sha384-QJHt

</body>
</html>
```

- Both templates inherit the single layout template. Layout.html is our parent template that will be included on every page, and complies with the inserted blocks unique to the other templates.