Project Template: Templates

In this guide we will:

- Store data using an array of objects
- Create Flask Templates

Prerequisites:

- Basic understanding of HTML and CSS.

  If you're unfamiliar with these terms these are some suggested tutorials:

  HTML
  https://www.w3schools.com/html/default.asp
  Key tutorial sections:
    - Introduction, Basic, Elements, Comments, Classes

  CSS
  https://www.w3schools.com/css/default.asp
    - Introduction, Syntax, How To (-> External Style Sheet).

Before we start

The current project is split into 3 different types of files.

1. Python files (flaskblog.py)
    - Creates the instance of flask.
    - Stores the data to be displayed.
    - Performs routing.
2. HTML files (blog.html, home.html, layout.html and register.html)
    - Handles the webpage's structure and contents.
3. CSS files (main.css)
    - Handles the webpage's styling.

Run the code in '**flaskblog.py'** and then make sure you can view both the web page and the code.

Step 1: Store data using an array of objects

Let's first have a look at the variable **posts** in **flaskblog.py**. All posts are stored and accessed using this variable. The key is to understand its structure.

posts is an array (identifiable by the square brackets, […]) containing two objects (identifiable by the curly brackets, {…}), and each object represents a post.

Inside each object are keys and corresponding values (identifiable by *key: value*) which represent the details of each post. Notice that the objects have identical keys, i.e. username, title, content.

Exercises

1. Compare the data in posts to the content displayed on the home page. What do you notice?
   The details in the website's posts match those stored in the variable **posts**.

2. Change the username for the first post from James to Sam, then save your changes and refresh the webpage. What do you notice?
   By changing **posts** and refreshing the page, the contents of the website's posts change to match.

3. Add the commented code below to 'posts', then save your changes and refresh the webpage. What do you notice?
   By adding a new object to **posts**, it adds a new post to the home page.

This demonstrates that there is a direct link between the stored data and the displayed pages. Soon we will look at how this link is made.

Step 2: Create Flask templates

Before we work though the HTML files, please look at the following link:

http://flask.pocoo.org/docs/1.0/patterns/templateinheritance/

Firstly, you should notice some key differences between **layout.html** and the other HTML files.

1. **layout.html** does not get displayed as a webpage when performing routing in **flaskblog.py.**
2. The first line in all files except **layout.html** is:

   {% extends "layout.html" %}.

3. Only **layout.html** includes the necessary html tags, e.g. <!DOCTYPE html>, <html>, <head>, <body>, etc.

The reason for these differences is that **layout.html** is a base template. The key benefit of having a base template is that it removes the need to have duplicated code for common elements such as the nav bar across multiple HTML files. Yet, it still allows for specified areas to be customised for each page.

In our project, **blog.html**, **home.html** and **register.html** are called child templates and they use the same base template, **layout.html.** Many elements look the same between them, e.g. the navbar and the side bar, however, you'll notice there are still some differences between the pages.

In our project, the base template is set using the following code in the child template:

{% extends "layout.html" %}   index as child table

Within the base template are empty spaces marked by `{% block %}` tags.

These blocks are to be overwritten by the child.

Exercises

Find the following code in **layout.html:**

{% block content %}{% endblock %}

Observe how **blog.html**, **home.html** and **register.html** each use the {% block content %} tag and how it affects what is displayed on the different pages.

Step 3: Add styling

As previously mentioned, all displayed pages use a consistent style. The reason for this is that the base template, **layout.html**, links to the CSS file, **main.css,** which contains rules for styling the site.

The code for including the CSS file is as follows:

```
<link rel="stylesheet" type="text/css"
  href="{{ url_for('static', filename='main.css') }}">
```

Due to inheritance, all the child templates inherit these styling rules from the base template.

If you wanted to include an additional set of style rules for a specific page, you would need to create a new CSS file and include it in the child template.

.