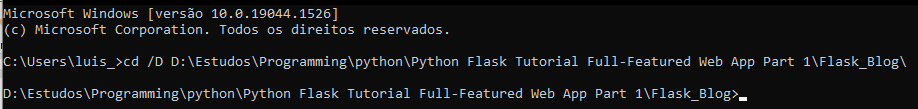
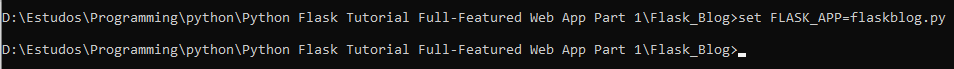
**Class 1/15: Getting Started**

CMD commands to start application:

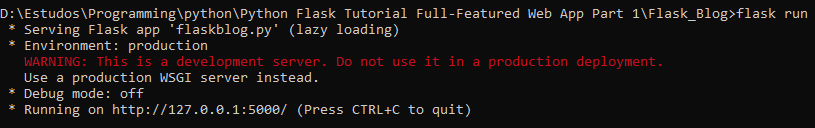
[1]



[2]



[3]



To open application on browser use the URL:



To refresh a browser page it's needed to reload the webserver via command line:

CRTL+C



redo start application commands

In order to avoid all this trouble whenever I want to reload the browser page we need to set another environment variable called FLASK\_DEBUG equal to 1 instead of FLASK\_APP="py file name"[2] and run flask[3] we can simply reload the browser page whenever we want to see a change

In order to accelerate this process, we can set the following if statement:

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

in this file and whenever we want to start the application we simply type on the CMD:

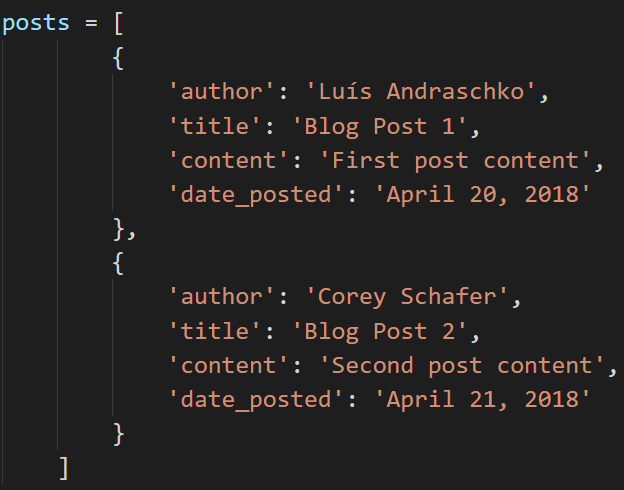
python flaskblog.py

**Class 2/15: Templates**

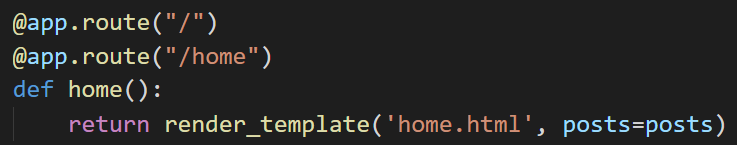
There is nothing stopping me from returning all my html code how It's been done in class 1/15, but that would be chaotic giving the size of the html codes. A better solution is to use templates. To use templates, we first need to create a templates directory within our project (same level as our .py). Put your html files inside templates directory, import render\_template function from flask and instead of returning "html code" we can return render\_template('home.html') for example.

**1) Passing variables to the template**

To pass a variable to the template we can create it in the .py file that contain the routes. For example, the following dictionary:



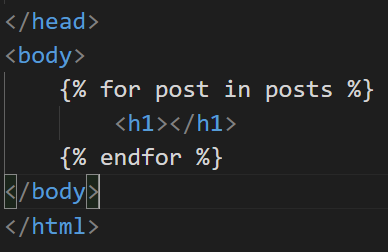
This dummy data can represent the return of a database call. To pass this data to our template we just need to pass in an argument with our data. To do this I can go to the definition of home function in our .py file and insert in the render\_template function arguments the following:



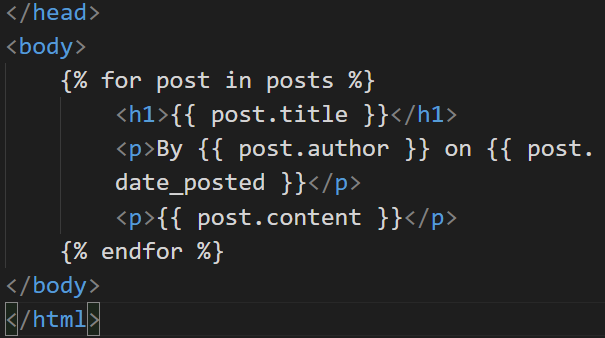
In this way, whatever the name of the variable we choose to pass in our data, that’s going to be the variable that we have access inside the template.

**2) Looping through data inside the template**

The templating engine that flask uses is called Jinja2 and It allows us to right code inside the templates. To do that we first need to open and close a code block in the template as follows:



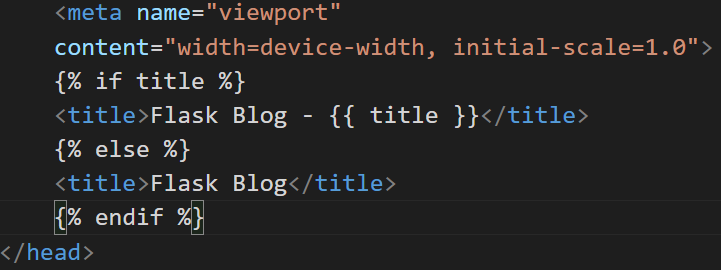
As you can see, in this way we are creating lines of html code by the repeating effect that the for loop has. Furthermore, to use the variable that we passed to the template we need to use the following syntax for jinja2 to work properly:



So, whenever accessing a variable inside a template the two curly braces are needed.

**2) Passing custom titles to html pages**

We can implement custom titles for the html with a code block containing an if statement in the head of our html as follows:

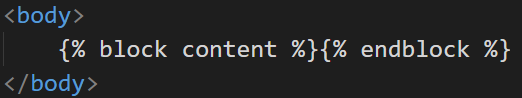


**3) Making a better design**

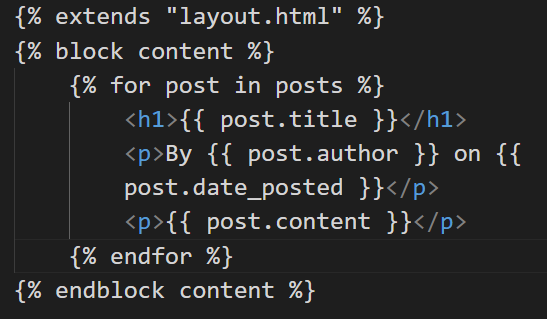
Commonly in programming we try to avoid repeating code, this way, whenever changes have to be made, and they will, we have to correct minimum amount of code. In our project so far, we already can notice that our html templates have a lot of repeating code. To avoid this, we will use a concept called template inheritance, by doing that we will create a standard template with an override block, by doing this instead of repeating the standard html code we can work with child templates of the standard and use the override block to custom the page. In this way most of our html files will have only what’s unique to them. To implement this, lets create a html file called layout.html and put all the shared code between home.html and about.html, as follows:



After doing this we’ll need to create a block, and a block is a section that the child templates can override. Notice that we’ll insert the block in the body of the html file, and that is because the uniqueness of the pages we’ll be in this section. The syntax for the block section is the following:



Content is just the name of the block, It could have been anything. Now that we have the proper standard html file, we have to leave in the other files just what’s unique to them. Furthermore, for the override to work we need to put a code block on the top of our html child file and use the extends key word and indicate inside “” the file which we want to extend from, and finally we wrap the unique code of the page in a block, for the home.html we’ll have:

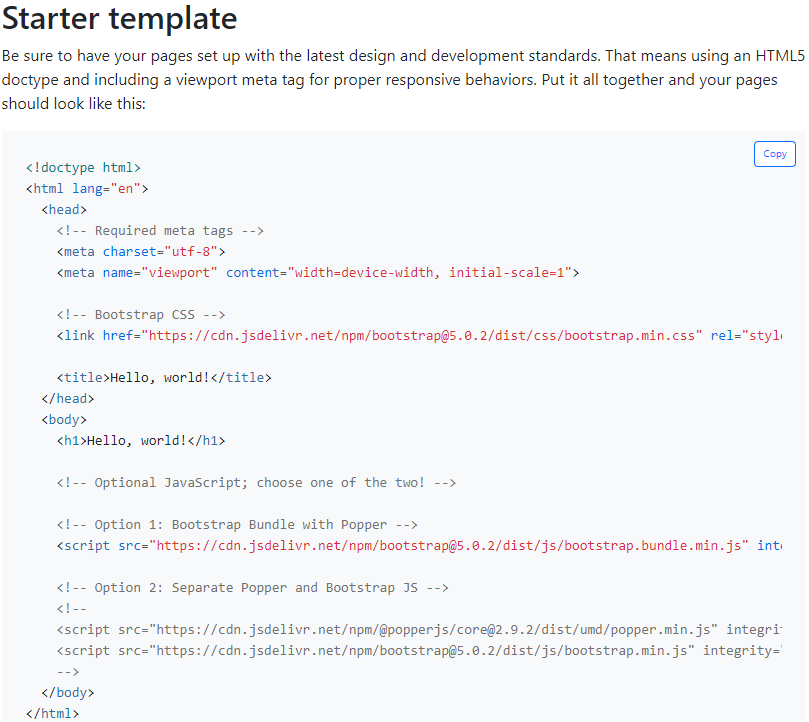


It’s important to note that the name of the block is not necessary in the endblock, but with multiple templates It’s a way to make the code more organized.

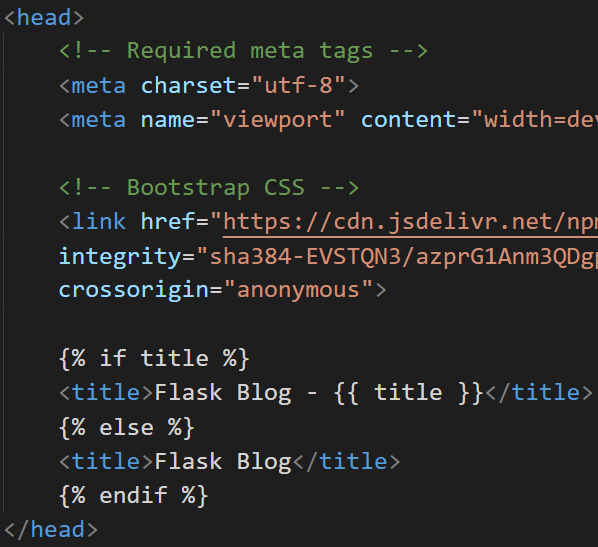
[OBS: CDN (Content Delivery Network]

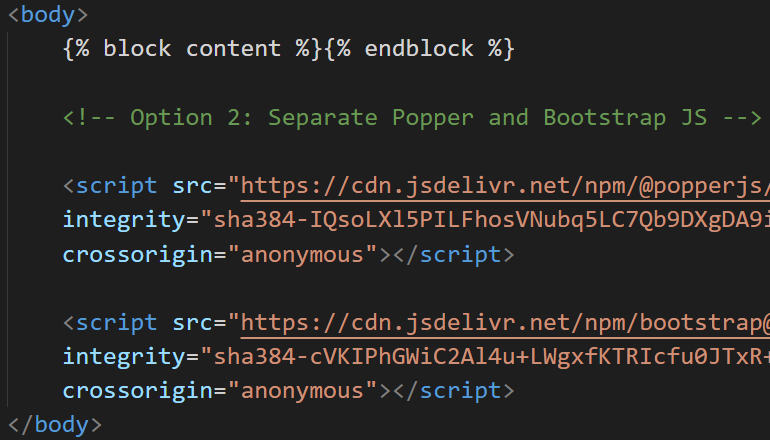
**4) Styling with bootstrap**

Bootstrap is the world’s most popular framework for building responsive, mobile-first sites, with jsDelivr and a template starter page. The use of bootstrap in this project is to avoid taking the focus out of flask. From the starter template we can start (<https://getbootstrap.com/docs/5.0/getting-started/introduction/> ).



Since all the CSS and JS used in these templates are from a CDN we don’t actually have to download anything. It’s important to say that If we weren’t using template inheritance we would have to apply this bootstrap or any other styling for that matter in each html file. That said, let’s apply this template to our layout.html:





To make this all work let’s use a bootstrap specific class “container” and put our block content inside a div of this class:

