**Server: Flask Routes.**

Pre-class work, Section 5A. (Approx. 2h50)

Videos can be found in learning central under this module > Learn Plus Recordings Example files associated with this content can be found in the .zip folder for this section.

Contents

[Initial brief reading (5m) 1](#_bookmark0)

[Session Introduction 1](#_bookmark1)

[Introduction (10m) 2](#_bookmark2)

[The Flask framework (10m) 2](#_bookmark3)

[Exercise (15min) 3](#_bookmark4)

[Routes lead to the execution of code ( 20m ) 4](#_bookmark5)

[Exercise (30m) 5](#_bookmark6)

[Processing a form GET(25m) 6](#_bookmark7)

[Processing a form POST(20m) 7](#_bookmark8)

[Doing something with the data (5m) 8](#_bookmark9)

[Exercise (20m) 8](#_bookmark10)

[Possible issues that break things 8](#_bookmark11)

[Extra Reading 9](#_bookmark12)

[Appendix 1: How to start the python flask server 9](#_bookmark13)

## Initial brief reading (5m):

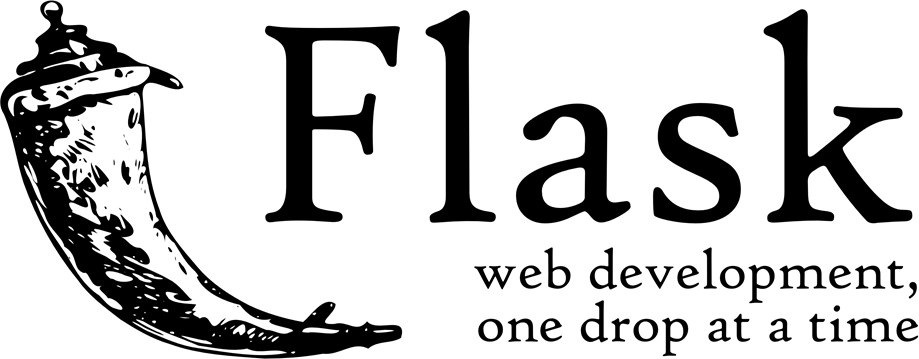
Briefly familiarise yourself with the Flask Documentation. <http://flask.pocoo.org/> (this will redirect to the new site <https://palletsprojects.com/p/flask/> ) i.e. have a look at the index, and take a 5min look at the “Quickstart”

## Session Introduction

### Associated video 5B\_ServerFlask.mp4

This session will allow you to create a server that will respond to different URL requests, different message types and extract data sent via forms. It is an introduction to the Flask server framework.

## Introduction (10m)

What is Flask?

* Flask is a python based micro framework
* A simple to use server which covers the basic principles that we need to cover.
* The principles are very similar to other frameworks such as

nodeJS or Django or even Laravel.

* <https://palletsprojects.com/p/flask/>

In a previous session we saw a couple of servers (0\_1\_pyserver and 1\_StaticFlaskServer) , one written in python which only served a single resource, and took a lot of code and the second which used a framework and just started the server to provide access to any resource in the static folder.

The point is that server frameworks can do a lot with some standard efficient coding. When a server receives a request it will process the URL and extract the route of the request. That is the directions to the resource. In flask, by default, any route starting with /static/ will serve the resource from within the folder static. E.g. http://localhost:5000/static/hello.html.

Recap: Starting the flask server

Create a virtual environment (pipenv shell –python 3.7) or 8 or ? Load the dependencies (pipenv install) if there is a pip lock file.

Run the server (py <Server>.py

## The Flask framework (10m)

To be read in conjunction with video **5B\_ServerFlask.mp4** To set up the Flask web application framework we need to write a few lines of code.

#server.py

from flask import Flask app = Flask( name )

…

if name == " main ": app.run(debug=True)

The first line is a comment stating that the file is server.py, it is not needed but helps with the session. We then import the Flask library code.

We need to create our flask app using library code from the flask module, so we call a constructor to create a Flask object. **app = Flask( name )**

Following this we would usually put code that creates a bespoke server.

Next, we need to run the server - to run the app we can either: Run the app from code:

**if name == " main ":** Is about importing from other modules. If this file (server.py) is being imported from another module then it is not set up correctly

and there is likely to be another server running so don’t run. This may be covered in the computational thinking module - for now just use these lines from the documentation.

**app.run(debug=True)** calls the run method from the app object which starts the server. Now we just need to run the code – **py server.py**

Or use python’s flask command:

For this, we need to tell the system which file contains the app on the windows cmd prompt this is done by **set FLASK\_APP=server.py** we can then run the app with the command **flask run**

Using this method we do not need the last 2 lines in the server file. **if**  **name** **== "** **main** **":** and **app.run(debug=True)**

These methods will allow the server to listen internally on the machine (localhost or IP 127.0.0.1) To allow external connections you need to specify the host = 0.0.0.0

In code, that means:

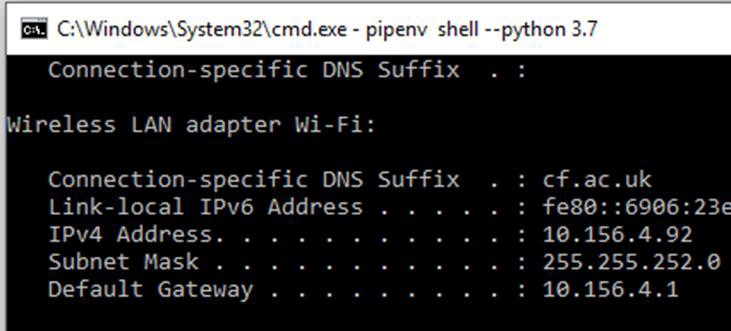
app.run(host='0.0.0.0', port=5000)

and using the flask command that means adding a switch to get **flask run --host=0.0.0.0**

Addition:

Default folder for static files is static, but this can be changed: app = Flask( name , static\_folder = 'web')

## Exercise (15min)

* Set up the 1\_StaticFlaskServer.py example
* Add a ‘static’ directory (if not there already)
* Add some of your pages to /static
* Request pages from a client on your machine 127.0.0.1:5000/static/?????
* Request pages using clients on your LAN (another pc or phone)

Open a new cmd window and type

**ipconfig**

Your IP address 

Note by default flask serves on port 5000

In this example use 10.156.4.92:5000/static/??????

**SHUT DOWN YOUR SERVER when done. (press ctrl +c in the command window)**

## Routes lead to the execution of code ( 20m )

So far we have set up a server that just serves static pages from a specified directory. The frame work has analysed the route e.g. **/static/hello.html** and that has led to the server executing code that retrieves the hello.html file and sending a response with the file contents.

But this is a very limited use of a server. A server framework can listen to requests that specify other routes other than those in the /static directory. Those routes can be used to trigger bespoke code.

You can create your own route in the flask framework by specifying a route and creating a function that will run when the route is requested.

@app.route("/hello") def myHello():

return "Hello World!"

*The @ Annotation: Calls framework code to set the function myHello as the function to be called when the server receives a request for the route /hello*

Look at the server code for the server **2\_ FlaskServer.py** ensure the /hello route is uncommented and **run the server**. Browse to **localhost:5000/hello** you should see a response message. This is not HTML but the browser will still display it. Also the code prints a message to the server console.

So… Routes lead to execution of code – this could be code to return file contents, code to return data or code to trigger an action (start motor to close curtains).

We are starting to get an idea of an Application Programming Interface (API) which is a set of routes with specified actions.

Redirect

from flask import Flask, **redirect**

**…**

@app.route("/redirect") def redirectToStatic():

print(“do some thing here”)

return **redirect**("/static/hello.html")

We may wish to execute code then return the contents of a file. In which case, following the code we wish to execute, we can use a function to redirect to framework code which returns the file contents.

**Add** this to the server file (or un-comment it) then re- start the server to try out the new route.

URL Variables

Some time we would like to use part of the URL as the route and part of it as a variable within the route. i.e. process many different URLs within the same route code.

e.g. if I wanted to get access to a resource (data on a staff member) “Ian” maybe I could write a route /staff/Ian but then what about /staff/Alexia and /staff/Wendy and /staff/Louise? I don’t want to rewrite all those routes. It is

@app.route("/Staff/**<person>**", methods=['GET']) def returnPerson(**person=None**):

if request.method == 'GET':

print("getting person "+ staff[**person**])

…

better to write the route

/staff/<person> and use person as a variable within the route’s code.

The URL variable is in angle brackets (or carrots) and we can

set default values for the variables in the function arguments (e.g. **person=None**).

If a URL variable is optional then we can use multiple routes, and set defaults for values that may not be there.

@app.route("/pathVars/<name")

@app.route("/pathVars/<name>/<surname>") def pathVars(name, surname="Doe"):

**Add** the pathVars route to the server file (or un-comment it) then re-start the server. Try-out some applicable URLs.

## Exercise (30m)

Open 2\_FlaskServer.py and complete the exercises at the bottom of the file.

1. Add a new route: /goodbye
   * Returns the text “goodbye”
2. Add a route: /goodbye/<name>
   * Return “goodbye Ian” where Ian is the name in the url.
3. Add a new route: /time
   * Returns the current time - Hint: import datetime

current\_time = datetime.datetime.now() current\_time.isoformat()

### Extras:

1. Add a new route: /hitCounter
   * Returns "Hello World! - from Flask server 1, you are visitor no. [5]”
   * Hint:

Use a global variable.

1. Add a new route to print the IP address of user to the server console **and** redirect to a static page of your choosing.
   * Hint:

from flask import request print(request.environ['REMOTE\_ADDR'])

# Processing a form GET(25m):

Do you remember your forms… There was an action attribute that we ignored. Look at

### static/form.html

<form action=" /**HomeForm**" method= "get" id="myForm" name="myForm">

<label>first name:

<input type = "text" **name = "firstname"**><br>

</label>

…

</form>

If you **browse** to the form page,fill in the form and click submit you will get something like: http://localhost:5000/HomeForm?firstname=Ian&text=werwerwe

in the URL bar. The data following the ? is the form data. (and probably an error as the form server is not running)

We need to add code to process the form in the server. The action (in the HTML file) points to

/HomeForm so the browser creates a request to /HomeForm so we need to create a route for

/HomeForm. In this route we need to specify which request methods we will accept (e.g. GET, POST or PUT etc. ) and then extract the form data from the request so we can use it tin the route code.

from flask import Flask, redirect, **request**

**…**

@app.route("/**HomeForm**", methods=[‘GET']) def form():

if **request.method** == 'GET':

print(**request.args.get**(**'firstname**')) return "Hello World!"

We can import more library functionality (request) which is an object that the flask server populates and makes accessible

to the route’s code. If the request method is a GET message (we find this out by examining the request object’s method variable - if **request.method** == 'GET': ), we can use a method within the request object’s args object to extract the specific request arguments

**request.args.get**(**'firstname**').

Look at 3\_FormServer.py run this server and repeat the process of requesting / filling in / and submitting the form. Look at the console to see the firstname printed out.

### Quick exercise:

1. Add a surname input field to **form.html**
2. Add to **3\_FormServer.py** to print the surname to the server console when the form is submitted via a GET request.

### Another Quick exercise:

Remember the Forms…..

* + Install "Screenreader for Google Chrome"
  + <https://chrome.google.com/webstore/category/extensions>
  + Search for Screen Reader for Chrome
  + Add to chrome
* Tryout the screen reader by clicking on different parts of the text
  + This will only work from the served page – not the file!
* Exchange the label tags for div tags. What happens with the screen reader?
* Remove the div tags. What happens with the screen reader?

# Processing a form POST(20m):

Another way of making requests is to sent a POST message. Post messages send the form data within the message contents rather than the URL. So the URL of a post to /HomeForm will look like this http://127.0.0.1:5000/HomeForm. The only difference in the HTML Form is the method attribute needs to be post.

<form action=" /**HomeForm**" method= "**post**" id="myForm" name="myForm">

<label>first name:

<input type = "text" **name = "firstname"**><br>

</label>

…

</form>

In the server file route code we can change the GET to POST where we specify which methods to accept or add the ability to handle both [‘GET’, ‘POST’]. To retrieve input parameters from the form we use a different technique. The request has a form data structure (dictionary) with key value pairs, the input name being used as the key. We access form data with this **request.form['firstname']**.

@app.route("/HomeForm", methods=['POST'])

def form():

if request.method == **'POST**':

print(**request.form['firstname']**) return "Hello World!"

Change form.html so it uses a post method to submit and re submit the form. Inspect the messages to see the parameters submitted using the network tab. (from Client Server session).

### Quick exercise:

Add to **3\_FormServer.py** to print the surname to the

server console when the form is submitted via a **POST** request.

Hint : you will also need to change **form.html** if you have not already done so.

# Doing something with the data (5m).

names = [] surnames = []

…

…

if request.method == 'POST':

name = request.form['firstname'] surname = request.form['lastname'] names.append(name)

surnames.append(surname)

As mentioned, We can now do anything with the data so a simple example is given here. (Storing in a list) – see (4\_FormServer.py)

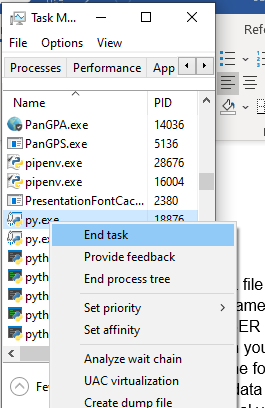
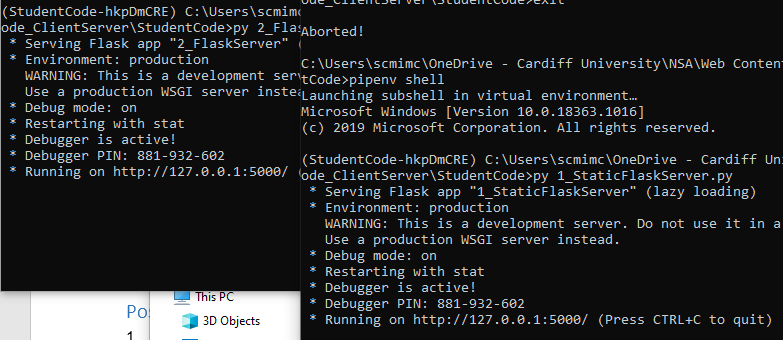
# Exercise (20m)

### Personal Data Collection

1. Create an HTML file with a form element.
   * Collect: name, date of birth, email address and postal address.
   * REMEMBER - Place it in the static folder
2. Create a route in your server to process the form:
   * Print all the form data on the server console.
3. Using a python data structure, store all the form data from each submission.
   * Hint: Global variables
   * Create another route to print all stored data to the server console

# Possible issues that break things :

1. **Wrong server** – you are not running the server you think – check all command windows and shut down all servers (ctrl +c keep pressing if it fails at first)



2. **2 servers running** – if you do not terminate the server properly a command window may shut and leave the server running.

The first server will be allocated port 5000 so the second server is useless. You may need to look at the task manager to end any other python processes.

1. **Wrong route** – check you are requesting the correct route. Look at the inspect tools network tap to verify.
2. **Virtual environment within a virtual environment** – if you have set pip files with in a directory with pip files the virtual environment can get very confused. Be carefull. You can delete the pipfile and pipfile.lock but you will need to re start the virtual environment again and reinstall the dependencies manually.
3. **When it still does not work** shut down the pc and start again – there may be other servers running that you can’t find.

## Extra Reading:

Read <https://termly.io/resources/articles/gdpr-for-dummies/> GDPR will be covered more in the programme but this is a starter.

# Appendix 1: How to start the python flask server.

* + Open a command prompt in the session files directory (the directory with the server in it),
  + start a virtual environment (**pipenv shell**),
  + install dependencies (**pipenv install**) if this does not work try **pipenv install flask**
  + run the python server file e.g. server.py (**py server.py**)
  + in flask, static files are server from a static directory. So to load a static hello.html page, go to your browser and type **http://localhost:5000/static/hello.html**
  + All being well you should see your page.

### Terminate this program using ctrl +c and shut down the cmd window.

Appendix 2: Frequent issues

**Issue)** The server does not seem to have updated – seems to be running the same code as before you changed it, saved it and restarted the server.

**Solution)** The most likely reason is that you did not shutdown the previous server correctly and the port (5000) is still associated with the previously running server instance.

To fix this, you will need to completely shut down all the server instances. The easiest way to do this is to open the task manager, click on the details tab order the processes by name and then end all of the python processes. (right click on process > end task).

