Flask 1:
Essentials

Lab Objectives

Objectives of these exercises are to start developing websites using Flask, by:

- implementing *'Hello, World!'* starter application;
- creating templates and specifying routing;
- starting to work on navigation, styling, organising a proper project structure;
- pushing code to a remote git repository.

PRELIMINARIES

These are relevant to all labs:

- This exercise is not assessed.

- You should attempt completing all tasks, and in the order they are given.

- The instructions are written for Windows' `CMD`. If you are using a different operating system, you need to adapts the commands accordingly. Some suggestion for Unix-based shell `bash` are provided, but these are non-exhaustive.

- <mark>IMPORTANT!!</mark> The exercises will give you some basic understanding of how to develop a website in Flask. To get more comprehensive understanding of how this 'stuff' works, it is strongly advised that you do additional, independent learning, e.g. by going though the lecture notes, recommended resources (documentation, books and other suggested resources), as well as by completing additional tutorials of your choice. These are given throughout and at the end of the document. However, note that these are just suggestions and the list is non-exhaustive! There are lots of other various resources and tutorials on the Web.

- **Snapshots of the state of the project** at various points of the Flask lab tasks completion are available on the School's GitLab, in **this repository**: https://git.cardiff.ac.uk/scmne/flask-labs ⧉. Use the hyperlinks in the 'Git Tags' column in `README.md` file to access a particular snapshot. You can view or download these to check your progress (and, at times, perhaps, to save your typing *from scratch*). However, please make sure you understand each line of the code!

- Whilst the code in the snapshots was extensively tested, it's possible some parts of it would require further modification to get it working on your machine. Pay particular attention to making sure all the necessary project dependencies are present in `requirements.txt`.

- Remember, the teaching team is here to help - just ask for advice. It is also okay to discuss the solutions with your peers (these exercises are not assessed!), however, make sure you understand everything by yourself.

- If you want to practice these exercises on your own machine, please make sure that the versions of the technologies and tools you have on your machine are compatible with these instructions and the COMSC systems setup. The versions of tools and systems used in these labs are specified above. You may use different versions, but you would need to adapt these instructions accordingly.

- Examples in these labs use particular tools and technologies, e.g. `python shell`, `sqlite` CLI (command line interface), etc. You do not have to use these - use whatever alternatives your are familiar or happy with. However, it's worth noting that learning how to use CLIs (as opposed to a GUI) is a valuable skill for a programmer.

- If you are completing these exercises on a non-university network, **VPN connection might be required to connect to some services and servers**, (e.g. OpenShift, MySQL). For information on how to set up a University VPN connection see COMSC documentation: go to `https://docs.cs.cf.ac.uk/notes/` and select the 'VPN' tag.

- If you copy the code directly from these instructions, make sure to check the syntax is correct, e.g. tabbing. The code displayed in the instructions has some characters, which, when pasted, will throw compile errors, e.g. symbols indicating newline used for fitting long lines of code on the page, or other non-visible characters used by PDF.

Technologies and Tools

The following technologies and tools are used in these labs:
- ◇ Python (3.9.6)
- ◇ Flask (2.0.2)
- ◇ SQLite (3.36.0)
- ◇ git on COMSC's GitLab `https://git.cardiff.ac.uk/` (not the commercial GitLab server!)
- ◇ COMSC's OpenShift - `https://console.openshift.cs.cf.ac.uk/`
- ◇ COMSC's MySQL server (csmysql.cs.cf.ac.uk) with phpMyAdmin (`https://phpmyadmin.cs.cf.ac.uk/`)
- ◇ Other libraries and packages, appropriate to specific tasks - these will be indicated as we progress through the lab worksheets.
- ◇ Optional/alternative tools and technologies (e.g. MySQL) will be suggested, but constitute independent learning.

Abbreviations used in the instructions

- ▪ `CLI` – Command Line Interface
- ▪ `dir` – directory (folder)
- ▪ `db` – database
- ▪ `NB` – nota bene
- ▪ `repo` – repository
- ▪ `venv` – virtual environment.

INTIAL SETUP

1. Create the project directory `<YOUR_STUDENT_NO>-flask-labs` (e.g. `123456-flask-labs`) in a suitable location on your hard drive.

2. Create the **virtual environment**:

   > `py -m venv venv` (1)

   activate it:

   > `venv\Scripts\activate` (2)

   and then install flask:

   > `pip install flask` (3) (4)

   NB: if all the dependencies needed for the project are known and specified in the `requirements.txt` file, you can install these recursively, using the following command:

   > `pip install -r requirements.txt` (5)

   NB: Make sure `requirement.txt` is in the current working directory. (Typically, this file is placed in the project's root directory.) See `README.md` on GitLab or the lecture notes for details on how to export the project dependencies into `requirements.txt` file.

---

(1) If this command does not work, try: `python -m venv venv` or `python3 -m venv venv`
(2) On UNIX the command is: `source venv/bin/activate`. Use `deactivate` command for switching it off.
(3) Other libraries necessary for your project are installed using the same command, you just need to replace the package/library name with the one you need.
(4) If you are getting an error message when you use `pip` to install a python package, you might need to use `--user` option, i.e. `pip install --user PACKAGE`.
(5) You might need prefix this command with `py -m`i.e the command will be: `py -m pip install LIBRARY`.

STARTER APPLICATION: "Hello, World!"

Let us first create a simple **'Hello, World!'** application.

3. Create **hello.py** file in the project directory with the following content:

```python
from  flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, World!'
```

4. Go back to command prompt:

   (a) Enable the debug mode [6] [7]:

   ```
   > set FLASK_ENV=development
   ```

   (b) Tell Flask where to find your app:

   ```
   > set FLASK_APP=hello
   ```

   (c) Start the development server:

   ```
   > flask run
   ```

5. Go to: **127.0.0.1:5000** to check that the web page displays your '**Hello, World!**' message.

---

[6] **Note: whilst the debug mode is useful in development environment, it should be switched off when you deploy your website on the server − as it presents a major security risk!**
[7] On **Unix**, instead of **set** use **export ...**

TEMPLATING

At the moment, our page is just a very basic one, without any styling. To fix this, we can use Flask's templating functionality. We also need to do various other changes to start separating the application logic from presentation (remember MVC!).

6. Create dir `templates`, and within it two files `home.html` and `layout.html`.

   - `layout.html` is a template used for presenting information on our website in a consistent manner across the whole of the website. It will also be used to contain the website navigation (header, footer, side bar, etc.).

   - `home.html` inherits from `layout.html`, and is used for specifying what data we want to display on this web page.

   (a) In `layout.html` input the following code:

   ```html
   <!DOCTYPE html>
   <html>
   <head>
     <meta charset="utf-8">
     <title>My Wonderful Site</title>
   </head>

   <body>
     <div id="header">
     </div>

     <div id="content">
       {% block content %}
       {% endblock %}
     </div>
   </body>
   </html>
   ```

   (b) And in `home.html`:

   ```html
   {% extends "layout.html" %}
   {% block content %}
     <h1>Hello, World!</h1>
   {% endblock content %}
   ```

7. We have already specified the routing for our home page (Task 3). We now need to modify **hello.py** to tell the server to render the templates we are using.

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

@app.route("/")
@app.route("/home")
def home():
    return render_template('home.html', title='Home')
```

Note: Along with importing **render_template**, we also need to import **url_for** (line 1) to avoid handling of URLs manually (e.g. if we change a root URL we would need to change all templates/web page this URL is present).

8. Modify **home.html** by formatting the 'Hello, World!' sentence as **<h1>**. Check the output.

9. We can also tell the server where to look for any other pages we want to add to the website – by adding routing information for another page, **about.html** to **hello.py**.

   (a) Add the following to **hello.py**

   ```python
   @app.route("/about")
   def about():
       return render_template('about.html', title='About Me')
   ```

   (b) Create a new file called **about.html** in the templates folder with the following contents:

   ```
   {% extends "layout.html" %}
   {% block content %}
       <h1>This is About Me page</h1>
   {% endblock content %}
   ```

10. Restart the development server to check you can access the 'About Me' page by going to **127.0.0.1:5000/about**.

    NB: You might need to reset **FLASK_ENV** and **FLASK_APP** variables to start the server - see Item 4.

11. Let's add some navigation – by modifying `layout.html`:

    (a) Add links to the two pages we created, using `<a href="..">`. It is advisable to use Flask's `url_for` to avoid having to change URLs throughout the application, e.g.

    ```
    <a href="{{ url_for('home') }}">Home</a>&nbsp
    <a href="{{ url_for('about') }}">About Me</a>
    ```

12. Let's now add some styling to our web pages.

    (a) Create dir `static`, which is used for all 'static' files (e.g. CSS, JavaScript, images). Within this dir create a file `style.css` - to be used for styling our web pages.

    (b) To get Flask serve the static files we need to tell it about the location of static files (i.e. `static` dir) to `hello.py`:

    ```
    app = Flask(__name__, static_folder="static")
    ```

    (c) Modify `static/style.css` to change how you page looks like, e.g. change colour for the links, background colour for the page, etc. Check the result.

    Hint: you will need to add `<link..>` to `layout.html` in the form of:
    ```
    <link rel=stylesheet type=text/css href="{{ url_for('static',filename='style.css')
    ↪  }}">
    ```

Project Organisation

Unlike some other web development frameworks, Flask doesn't require you to have a specific directory structure for your project. It is possible to have your application in just one file. This can be appropriate for a small project. However, such flat structure of a project might not be desirable or indeed a good practice for a project that requires a substantial number of components, and in particular, if some components can be reused in different projects.

Therefore, before we start looking into implementation of essential functionality (database, etc.), we need to re-organise the project to allow for adding various packages – as well as making our project looking more professional!

13. **NB:** if you deactivated your virtual environment, you need to re-initialise and re-activate it (see Task 2 above).

14. Add a new file, `wsgi.py`, in the project's root dir (i.e. at the same level as `hello.py`), which contains the following code:

```python
from blog import app


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

15. Still, in your project's root directory (e.g. `123456-flask-labs`), create a sub-directory called `blog`.

16. In the `blog` dir:

    (a) Create `__init__.py`:

```python
from flask import Flask

app = Flask(__name__)
app.config['SECRET_KEY'] = '<paste SECRET_KEY here>'

from blog import routes
```

    Note: the last import should be the last line in the file.

    (b) Create `SECRET_KEY` [8]:

```python
> python
>>> import os
>>> os.urandom(24).hex()
```

    and then paste it into `__init__.py` file.

---

[8] Needed for 'sessions' later - see sections 'Sessions' on `https://flask.palletsprojects.com/en/2.0.x/quickstart` for more information.

(c) Create **routes.py**:

   i. Copy imports of **Flask**, **render_template** and **url_for**, and then add a new import - of the **app** object:

```
...
from blog import app
...
```

   ii. Copy and paste the routing information from **hello.py** to this file, i.e. the two routes for **home** and **about** (see Tasks 7 and 9).

(d) Move the **templates** and **static** directories you created earlier in the lab into the **blog** sub-directory (Tasks 6 and 12a, respectively).

17. The project directory structure should look like this:

```
./flask-labs
|   +\---blog
|   |   __init__.py
|   |   routes.py
|   |
|   +\---static
|   |       style.css
|   |
|   \---templates
|           about.html
|           home.html
|           layout.html
|  requirements.txt
|
\---venv
|  wsgi.py
```

Note: **hello.py** file is not needed anymore and can be removed.

18. Reset **FLASK_APP** variable to point to **wsgi**,i.e.

```
> set FLASK_APP=wsgi
```

Restart the server and check everything is working.

For this task, we will set up a remote `git` repository, so that you can push your local code to it.

NB: The instructions in this section assume that you have already added an SSH Key to your GitLab account. If you have not, you need to do that first. Follow the instructions, given Sec **'Appendix A: SSH Key pair and GitLab'** at the end of this document.

NB: You might need VPN connection established before you can connect to the School's GitLab server.

19. Create an empty project on GitLab:

    (a) Click on `New Project` button and then select `Create Blank project`.

    (b) Give your project (repo) an appropriate and meaningful name, such as: `YOUR_STUDENT_NO-flask-labs`, e.g. `1234567-flask-labs`.

    (c) For **Visibility Level** option, select `Private`.

    (d) Make sure to **deselect** the check button for **Project Configuration** option - we do not want to initialise this repo as we will want to push code from the existing repository.

    (e) Click on `Create Project`, and you will be presented with the 'front page' of your newly created repo. This page contains useful **Command line instructions**, In particular, we are interested in:

        - **Git global setup** section, which you can use for setting up git on your machine;
        - **Push an existing folder** section, which you can use for pushing your local project dir to GitLab.

20. Create a `.gitignore` file to specify which files and dirs you do not want to push to the remote repo, such as all `.pyc` files, `venv` dir, any files whose name starts with a specific characters, e.g. `logs*`. You can see an example of .gitignore file in the repository which demonstrates possible solutions to the Flask Labs (https://git.cardiff.ac.uk/scmne/flask-labs).

    NB: your `.gitignore` file needs to be staged and committed like any other file you want to push from your local repo to the remote.

    NB: If you already pushed an unwanted file to the remote repo before creating the `.gitignore` file, you would need to untrack the unwanted file first, using:

    ```
    git rm --cached FILENAME
    ```

21. Using the instructions in **Push an existing folder** section, put your project dir under `git` control, stage, commit and push the project files to the remote.

> ⚠ **Git doesn't accept blank spaces in file and directory names, and these need to be properly 'escaped'. To avoid issues it's suggested naming all files and directories without any blank spaces; use the hyphen and/or underscore characters instead.**

Useful resources relevant to all labs	
Snapshots demonstrating code at various points of lab tasks completion:	https://git.cardiff.ac.uk/scmne/flask-labs ⬀
Flask Website:	https://flask.palletsprojects.com/en/2.0.x/ ⬀
Flask Quickstart:	https://flask.palletsprojects.com/en/2.0.x/quickstart/ ⬀
Flask Tutorial:	https://flask.palletsprojects.com/en/2.0.x/tutorial/ ⬀
Books on Flask available in the library *(in no particular order!)*	Gaspar, D. (2018). Mastering Flask Web Development - Second Edition. Packt Publishing. DuPlain, R. (2013). Instant Flask web development. Birmingham, England: Packt Publishing Aggarwal, S. (2019). Flask framework cookbook : over 80 proven recipes and techniques for python web development with flask (2nd ed.). Birmingham; Mumbai: Packt Publishing Ltd. Grinberg, M. (2018). Flask web development: developing web applications with python. O'Reilly Media, Inc.
Free book on git: S. Chacon, B. Straub 'For Git'	https://git-scm.com/book/en/v2 ⬀
Video tutorials on Flask you might find useful *(not necessarily the latest Flask version; and again,in no particular order)*	[1], [2], [3] (these links are to the first lessons in a series, locate an episode relevant to a particular topic).
A number of cheat sheets could be found on the web.	

Passwordless login using Secure Shell (SSH) is required by some of our servers, e.g. for pusing your local repo to the remote on GitLab.

You can create your SSH Key pair using a number of ways. Here we will look at how you can do it: (a) using `git-gui`, and (b) via a terminal (e.g. Windows `cmd`).

(a) Using `git-gui`:

   i. Go to the command line prompt `cmd`

   ii. type `git-gui`. This will bring up the UI for git.

   iii. Go to **Help** -> **Show SSH Key**.

   iv. Click **Generate Key**.

   v. Enter passphrase (or just click OK if you do not want to use a password - I normally don't bother as I would most likely forget it).

   vi. The public and private keys (`id_rsa.pub` and `id_rsa`, respectively) will be created in your `<YOUR_NAME>/.ssh` dir [9].

   vii. Open the file containing your public key, i.e. `id_rsa.pub`, and copy the whole line, including `ssh-rsa ...`.

   viii. Go to your GitLab account, and then -> your **account icon** (top right hand side) -> **Edit Profile** -> SSH Keys (side bar on the left).

   ix. Paste your complete public key into the **Key** textbox. Give it an appropriate title and expiry date *(optional)*, and click on **Add key**.

(b) Using a terminal:

   i. Type:

```
ssh-keygen -t rsa -b 4096
```

   ii. Repeat the steps *(a.v)* to *(a.ix)* above.

[9] This is a hidden dir, so if you don't see it you might need to enable displaying of hidden files in Windows File Explorer. Alternatively, open your home dir with Atom - Atoms shows all files including the hidden ones by default.