N02a_Flask_Intro

In this chapter, students will learn and understand:

- Use the `flask.Flask()` constructor to create a Flask application
- Explain what a decorator is
- Use the `flask.Flask.route()` decorator to associate functions with HTTP paths
- Use `flask.Flask.run()` to start a Flask application
- Describe the benefits of using the Flask framework

Section 1: Creating Directory Structure

Before getting started, you will need to create the directories needed for this application.

There are some conventions in the storage and retrieval of files used in a Flask web application. The structure below will help you in organising files that are needed to run the application. When Flask object methods are called, files are retrieved from and stored into these specific directories.

Each time you create a new file, make sure to save it into the appropriate directory.

```
+/app
    +- /static
        +- /css
            +- style_sheet.css
            +- ...
        +- /images
            +- my_image.png
            +- ...
    +- /templates
        +- index.html
        +- about.html
        +- ...
    +- app.py
    +- db.sqlite3
    +- ...
```

- ***app*** directory will store the application package such as Python (.py) files and database files (.sqlite3).
- ***static*** subdirectory stores static files such as images (.png) and Cascading Style Sheets (.css).
- ***templates*** subdirectory stores template files (.html, .htm) that will use the Jinja template engine.

Create a directory structure according to what is shown below:

```
+/app
    +- /static
    +- /templates
```

Section 2: Creating and Testing a Flask application

The first web application that you will create will display text and values that are entered into the address bar of the browser onto a webpage.

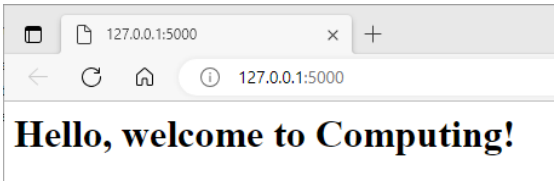
1. To test your flask installation, launch Python IDLE and create a file called **app.py** with the following code. Save it in the app folder.

```
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route('/')
5  def index():
6      return "<h1>Hello, welcome to Computing!</h1>"
7
8  if __name__ == '__main__':
9     app.run(debug=True)
```

2. Run app.py

3. Open your web browser and type http://localhost:5000/ or http://127.0.0.1:5000

You should see the following if your web application is successfully run.



Concurrently, you will also see the following being displayed on the Python shell. This means that the server is running and triggers an auto-reload of the files needed.

```
>>>
 RESTART: C:\Users\ somenameab \Desktop\4.Flask\3.Writing_your_first_web_applicat
ion\app.py
 * Restarting with stat
```

2.1 Introduction to Flask

So, how the flask application works? What do the following parts of the code actually mean?

```
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route('/')
5  def index():
6      return "<h1>Hello, welcome to Computing!</h1>"
7
8  if __name__ == '__main__':
9     app.run(debug=True)
```

Creates a Flask object and stores it in variable `app`

Configures the Flask object to run `index()` whenever the HTTP Request URI matches `'/'`

Tests whether the Python file is being run as a script

Line 1 imports the `Flask` class of the `flask` module. This is different to `from flask import *`, which imports all classes within the flask module.

Line 2 uses the `Flask` class to create an application instance, which takes in the `name` of the main module or package of the application as an argument.

Line 4 defines a decorator for the function below it. The decorator acts as a wrapper to an existing function by extending its original functionality, hence "decorating" the function. A decorator uses `@flask.Flask.route` to bind a URL to a Python function.

`flask.Flask.route(rule, options)`
The **`rule`** parameter represents URL binding with the function. The **`options`** are a list of parameters to be forwarded to the underlying **`rule`** object.

In this example, `@app.route` binds `'/'` to the `index` function. When a user navigates to `http://<somewebpage.com>/` on a web browser, it would trigger the index function to run on the web server and it will return its output on the webpage. Hence the user will see the text "**Hello, welcome to Computing!"** on the webpage.

Line 8 checks if the Python file is being run as the main script directly. If the script is imported from another script, the script will keep its given name. In this case, the script is directly executed and not imported. Therefore, `__name__` will be equal to `"__main__"`. This means the if conditional statement is satisfied, the `app.run()` method will be executed. This technique allows the programmer to have control over script's behaviour.

The `app.run()` method runs the application on the local server. By default, host is set to `127.0.0.1`, `port` is set to `5000` and `debug` is `False`.
For debugging purposes in a development environment, always set `debug` to `True` so that any errors will show on the browser.

`flask.Flask.run(host, port, debug, options)`

`host`
Hostname to listen on. Defaults to 127.0.0.1 (localhost).
 	
`port`
Defaults to 5000
 	
`debug`
Defaults to false. If set to true, provides debug information
 	
`options`
To be forwarded to underlying Werkzeug server

2.2 What is a decorator?

Function decorators (line 4) are simply wrappers to existing functions (lines 5 – 6). They modify the behaviour of the code before and after a target function execution. It extends the original functionality without explicitly modifying it, thus "decorating" the index function shown below.

```
4 @app.route('/')
5 def index():
6     return "<h1>Hello, welcome to Computing!<<h1>"
```

You can think of a function decorator as a function that returns another function, which is another level of abstraction to the user of Flask!

```
def funct_decorator():
    def index():
        return "<h1>Hello, welcome to Computing!<<h1>"
    return index
```

Section 3: What is the Flask Framework?

What is a web framework?

It is a code library that simplifies the building of web applications by:

- Reusing code for common HTTP operations
- Allowing quick and scalable development of web applications

Why is Flask a good web framework choice?

- It is a Python microframework that includes a small robust core with basic functionality for web applications.
- Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine.
- Comes with customisable extension parts.
- Provides easy to use boilerplate code for beginners.

Pre-requisites:

- Python programming experience
- Knowledge of basic HTML and CSS
- Concepts of packages, modules, functions, object-oriented programming, data structures such as tuples and dictionaries.

Terminologies:

- **Uniform Resource Locator (URL):** A Uniform Resource Locator (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
- **Uniform Resource Identifier (URI)**: A Uniform Resource Identifier (URI) is a string of characters that unambiguously identifies a particular resource.
- Technically URL is a subset of URI, where URI **identifies** while URL **identifies** and **locates**. There is a lot of confusion caused by the vague definitions and very often we could use the two terms interchangeably.
- **Decorator**: a wrapper to an existing function by taking it in and extending its original functionality without explicitly modifying it, hence "decorating" it. It is like a function that returns another function.
- **Route**: binds a Python function to a URL

References:

- MOE Python Flask Web Application Starter Kit
- MOE Teacher Training 2018 - HTML and CSS
- Udemy: Python and Flask Bootcamp: Create Websites using Flask!
- Udemy: Build Responsive Real World Websites with HTML5 and CSS3