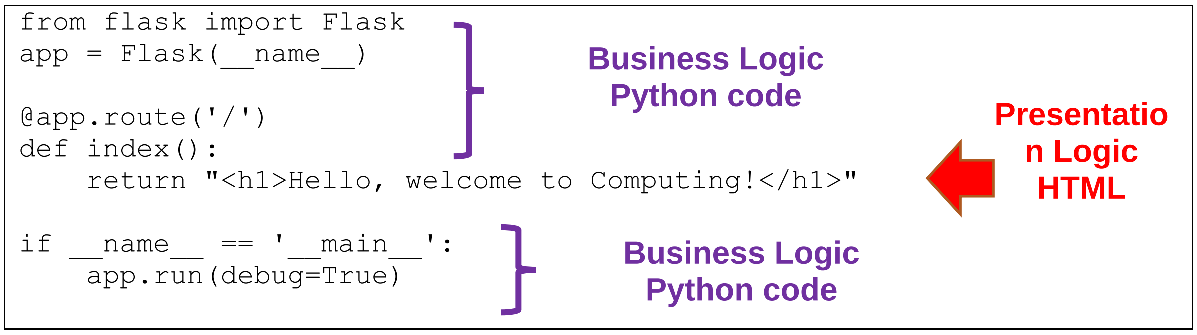
N02b_Render Templates

In this chapter, students will learn and understand:

- Describe the benefits of using render templates in Flask web applications
- Create a HTML template and save it to the templates directory
- Use `flask.render_template()` to simplify the sending of long HTML responses
- Use the `<link>` tag to associate a web page to a CSS style sheet
- Define multiple decorators for a single function
- Use trailing slash in a route
- Use converters to specify the data type for a variable

Section 1: Introduction

Previously what we did was to contain both business logic and the presentation logic within the same file as shown below. The business logic refers to the Python code, while the presentation logic refers to the HTML code.



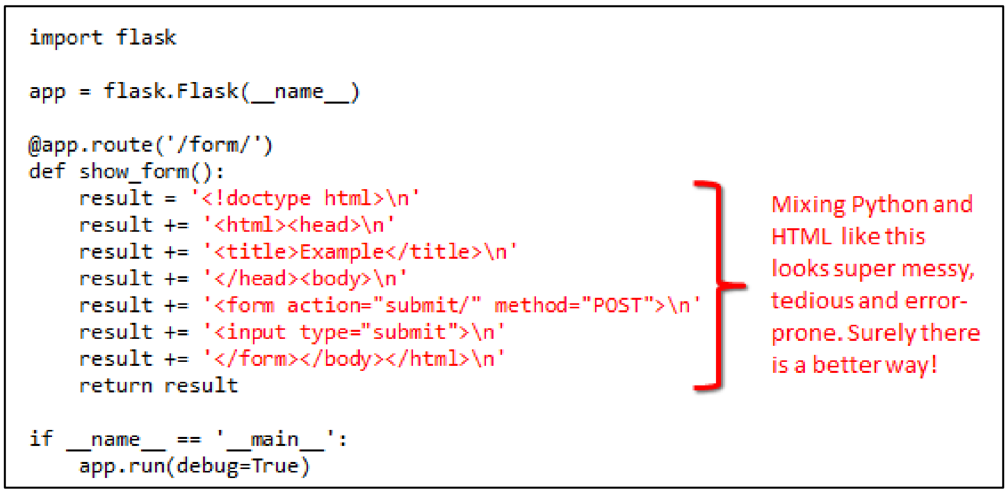
```
from flask import Flask
app = Flask(__name__)

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

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

The above code still looks manageable because it is relatively short.

However, what happens if your presentation logic grows and there are many lines of HTML code to process? What will it look like?



```
import flask

app = flask.Flask(__name__)

@app.route('/form/')
def show_form():
    result = '<!doctype html>\n'
    result += '<html><head>\n'
    result += '<title>Example</title>\n'
    result += '</head><body>\n'
    result += '<form action="submit/" method="POST">\n'
    result += '<input type="submit">\n'
    result += '</form></body></html>\n'
    return result

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

Mixing Python and HTML like this looks super messy, tedious and error-prone. Surely there is a better way!

Generating HTML content from within Python code is cumbersome, especially when variable data and Python language elements like conditionals or loops need to be processed and output. This would require frequent escaping from HTML.

Moving forward, we want to separate these two parts by extracting the presentation part and putting it into HTML templates and using the `render_template()` method inside the Python file to handle all these for us.

A template contains variables and/or expressions, which get replaced with values when a template is rendered; and tags, which control the logic of the template.

Hence we will make use of Flask's `render_template()` method that will take advantage of Jinja2's template engine to render a HTML file instead of returning hardcode HTML from the function.

Section 2: Rendering a Template

We will now create a class homepage web application and use `render_template` to render the HTML template. More functions will be added on and modified in the later lessons.

1. In templates directory, create a new file and save it as **index.html**.

```
<!doctype html>
<html>
<head>
    <title>Our Class Home Page</title>
</head>
<body>
    <h1>Hello, welcome to Computing!</h1>
</body>
</html>
```

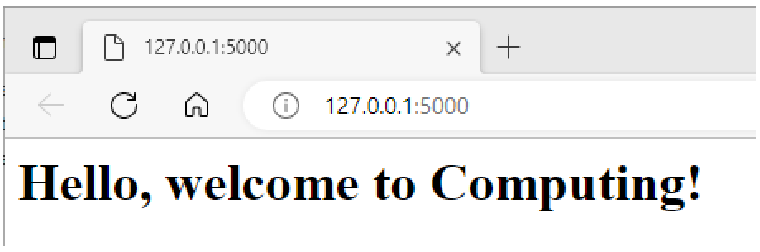
2. Open **app.py**. Modify lines 1 and 6. Here we are separating the presentation logic from the business logic by moving all HTML parts into a separate template file.

```
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route('/')
5  def index():
6      return render_template('index.html')
7
8  if __name__ == '__main__':
9     app.run(debug=True)
```

3. Run **app.py**.

4. 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.



Hello, welcome to Computing!

2.1 Using CSS for Presentation Style

CSS only describes the presentation of a web page and not its structure or contents. You can create a reference to a CSS file inside a HTML template using the `<link>` tag. We will now create a link to a CSS file inside index.html. Observe the difference in style before and after applying the CSS file.

1. Inside the **static** directory, create a new subdirectory called **css**.

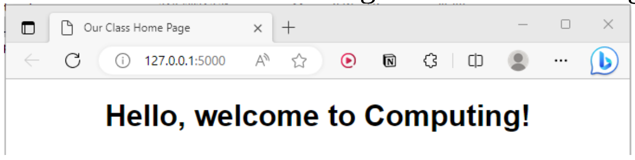
2. Create a new file and save it as **style.css** in the **css** subdirectory.

```
body {
    font-family: sans-serif;
}
h1 {
    text-align: center;
}
table {
    width: 400px;
}
th {
    text-align: right;
}
td {
    padding-left: 1em;
    padding-right: 1em;
}
td form {
    text-align: right;
}
```

3. Open **index.html**. Modify it such that it is linked to the CSS file.

```
<!doctype html>
<html>
<head>
    <link rel="stylesheet" href="/static/css/style.css">
    <title>Our Class Home Page</title>
</head>
<body>
    <h1>Hello, welcome to Computing!</h1>
</body>
</html>
```

4. Refresh your webpage. It should look as shown below, notice how the font is now sans-serif instead of Times New Roman and headings are now centre-aligned.



Hello, welcome to Computing!

5. To view page source, you can click right click on the webpage, then click **View Page Source**.

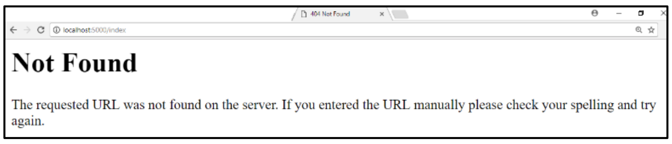
2.2 Binding Multiple Routes to a Single Function

Multiple routes can be bound to a single function by the use of decorators in the Python file. For example, it is by convention that both `'/'` and `'/index'` should lead to the same index function being called. However, in our example, only `'/'` will lead to the index function being called.

1. Run **app.py**

2. Type `http://localhost:5000/index` in your browser.

You should see a `HTTP 404` error as follows:



Not Found

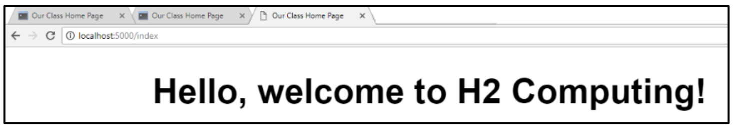
The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

3. In **app.py**, add a second decorator to your index function such that `'/index'` is also bound to the index function.

4. Run **app.py** again.

5. Test both `http://localhost:5000/` and `http://localhost:5000/index`

You should see the same output for both web addresses.



Hello, welcome to H2 Computing!

2.3 Using Trailing Slash

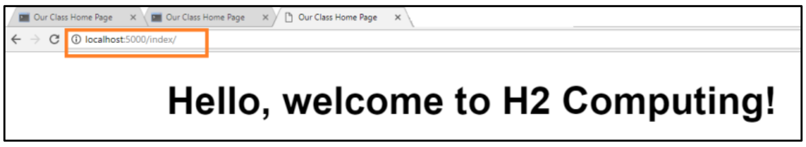
Now we have another issue. Both `'/'` and `'/index'` are bounded to the same index function. What happens when you try to access `http://localhost:5000/index/` in the browser? Note that when `'/index/'` is used, it returns a `HTTP 404` error because the Python script did not have any defined routes for `'/index/'`.

1. Open **app.py**.

2. Modify your code such that both `/index` and `/index/` will result in the same **index.html** webpage being rendered and displayed in the browser.

3. You need **not** define an additional route.

What you should see:



Hello, welcome to H2 Computing!

What you have just done is to use a trailing slash `/index/` which will result in a canonical URL. Hence both `/index` and `/index/` will return the same output.

Exercise 2.3

For each sentence, evaluate if it is **True** or **False**.

	1. Each function in a Python Flask web application must have at least one route.
	2. A function in a Python Flask web application can be bounded to multiple URLs.
	3. In a Python Flask web application, there can be multiple functions of the same name.
	4. The use of `@app.route('/myhomepage/')` will give two different views of when `/myhomepage` and `/myhomepage/` are entered into the browser.

2.4 Rendering A Template with a Variable

Flask comes packaged with a powerful template engine called `Jinja`, which uses delimiters to escape from HTML. This means that these expressions are not treated as HTML.

`{% ... %}` is used for expressions or logic such as conditionals and loops.
`{{ ... }}` is used for outputting the results of an expression or a variable to the end user.

The latter tag, when rendered, is replaced with a value, and is passed back as a response to the browser. More information on `{% ... %}` will be provided in the next task. In the below example, we will use `{{ ... }}` first.

1. Create and save a new HTML template called **about_us.html** in the templates directory. You may use the example below or add your own information about your class in this file.

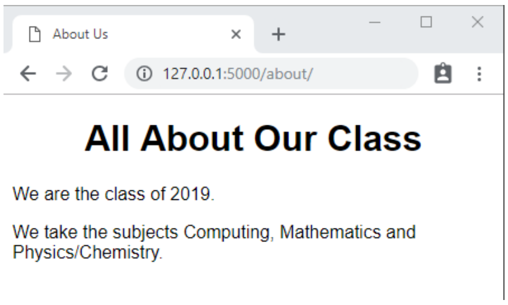
```
<!doctype html>
<html>
<head>
    <title>About Us</title>
    <link rel="stylesheet" href="/static/css/style.css">
</head>
<body>
    <h1>All About Our Class</h1>
    <p>We are the class of 2019.</p>
    <p>We take the subjects Computing, Mathematics and
Physics/Chemistry.</p>
</body>
</html>
```

2. Inside **app.py**, add a decorator that will bind the URL `'/about/'` to a function that will render the **about_us.html** template. You may use a suitable name for the function. An example is shown below.

```
@app.route('/about/')
def about_us():
    return render_template('about_us.html')
```

3. Run **app.py**.

4. Type `http://localhost:5000/about` in your browser. You should see the following:



All About Our Class

We are the class of 2019.

We take the subjects Computing, Mathematics and Physics/Chemistry.

Now we will modify the HTML template and the Python file to use a variable. This variable will be entered by the user in the address bar of the browser. It will be passed into the function as an argument. Flask's render template method will then pass this variable over to the template for display on the browser.

5. In **about_us.html**, add these lines.

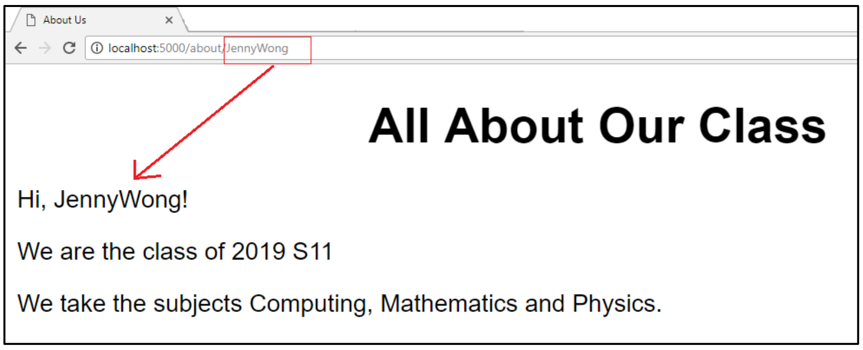
```
<!doctype html>
<html>
<head>
    <title>About Us</title>
    <link rel="stylesheet" href="/static/css/style.css">
</head>
<body>
    <h1>All About Our Class</h1>
    <p>Hi, {{ name }}!</p>
    <p>We are the class of 2019.</p>
    <p>We take the subjects Computing, Mathematics and
Physics/Chemistry.</p>
</body>
</html>
```

6. In **app.py**, modify your code such that:

- `@app.route` will include something like `'/about/<variable_name>'` where `<variable_name>` refers to the variable that is entered by the user.
- Modify your Python function such that it is passed in as an argument.
- Modify your `render_template` method such that it is passed in as an argument and assigned to the name variable, which was specified as `{{ name }}` in **about_us.html**.

7. Run **app.py**.

8. Type `http://localhost:5000/about/JennyWong` in your browser. You should see the following.



All About Our Class

Hi, JennyWong!

We are the class of 2019 S11

We take the subjects Computing, Mathematics and Physics.

The method `render_template()` will take in the template file as its first argument and key-value pairs as additional arguments that will represent actual values for variables referenced in the template.

When this method is called, it will read the template file and extract its content. If there are additional arguments defined, it will pass them over to the template for rendering. `{{ name }}` in **about_us.html** is a placeholder for a value that should be obtained at the point of rendering. The rendered response is then passed back to the browser.

2.5 Rendering A Template with Converters

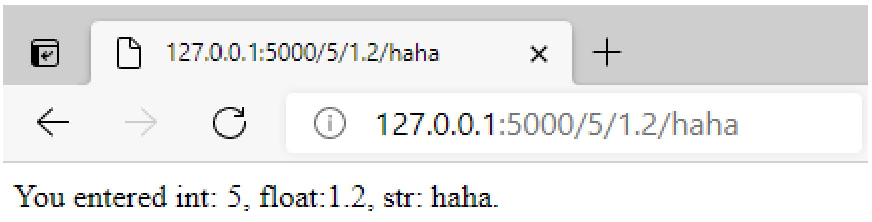
What you have just learned is how Flask handles variables that are entered by a user through the browser. You may also use a converter to specify the data type for the variable passed in using the syntax `<converter: variable_name>`.

Note that this file is used for testing purposes and will not be part of the class homepage web application.

1. In **app.py**, add the following code.

```
@app.route('/<int:i>/<float:f>/<s>')
def test_variables(i, f, s):
    return 'You entered int: {}, float:{}, str: {}.'.format(i, f, s)
```

2. Run the script again and test it out on a web browser.



```
You entered int: 5, float:1.2, str: haha.
```

3. Test again but now try entering 2 integer values. What happens?

4. Test again but now try entering 2 floating point values. What happens?

In summary, the types of converters that you need to know are described below.

Converter	Syntax	Description
`string`	`<s>`	accepts any text without a slash (the default)
`int`	`<int: i>`	accepts integers
`float`	`<float: f>`	accepts floating point values

Exercise 2.5

Modify **about_us.html** and **app.py** such that the webpage will display the **name** and **age** of the user. The user will enter both name and age at the same time by typing them into the address bar of the browser.

Terminologies:

- **Template:** a file that contains the text response, with placeholder variables for dynamic parts.
- **Rendering:** a process that replaces placeholder variables with actual values and returns a final response string.
- **render_template():** renders the `Jinja2` template with the given filename and arguments, and returns the response as a string.

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