CHAPTER 4
TEMPLATE BASIC 1

LEARNING OBJECTIVES

1. Students are able to analyze the purpose and use of templates in Flask by examining how they help separate application logic from the user interface.
2. Students are able to interpret and modify basic Flask templates that display data passed from the backend using variables.
3. Students are able to implement basic HTML templates in Flask by creating a simple web page that dynamically displays content using template variables.

CONCEPT AND SYNTAX

a) Template Basic

As a web framework, Flask uses templates in the form of HTML files to display web pages dynamically. This is possible because Flask utilises Jinja2 as its template engine. Jinja2 is a built-in templating system that allows you to insert variables, loops, and build simple logic and interfaces. This separation makes the code structure cleaner, tidier, and more manageable especially for medium to large-scale web applications.

The way Jinja2 is configured is:

- autoescaping is enabled for all templates ending in .html, .htm, .xml, .xhtml, and also .svg when using render_template().
- autoescaping is enabled for all strings when using render_template_string().
- templates have the ability to opt in/out of autoescaping with the {% autoescape %} tag.
- Flask inserts several global functions and helpers into the Jinja2 context, in addition to the existing values by default.

Syntax 4.1

```
from flask import Flask, render_template

Flask_objcet_name = Flask(__name__)
@Flask_objcet_name.route('/path')
def view_function_name():
   return render_template('template_name.html')
```

- **render_template** is the name of the function to render (display) the HTML template to the browser
- **view_function_name()** the name of the function that will execute render-template
- **template_name.html** is the html file that will be displayed in the browser

Example 4.1

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
```

```
def index():
    return render_template('index.html')
```

- **from flask import Flask, render_template** is Importing the Flask class to create the application, and the **render_template** function to display the HTML file.
- **app = Flask(__name__)** is Creating a Flask application object, with the module name as an argument.
- **@app.route('/')** is setting the root URL (**'/'**) and connecting it to the **index()** function.
- **def index():** is Defines the function that will be executed when the URL **'/'** is accessed.
- **return render_template('index.html')** is Display the **index.html** page from the templates/ folder.

As additional information, in using templates, HTML files are stored in the 'templates' folder as shown in the following folder structure:

In module:

```
/application.py
/templates
    /hello.html
```

In package:

```
/application
    /__init__.py
    /templates
        /hello.html
```

There are 3 types of delimeters in using Jinja templates. By default Jinja delimeters are written as follows:

1. **{%...%}** for statements

Example 4.2

```
<ul>
{% for fruit in fruits %}
    <li>{{ fruit }}</li>
{% endfor %}
</ul>
```

- The code inside **{%...%}** is used to loop through the list fruits. The list of fruits will be printed inside the <ul> tag.

2. **{{...}}** for expression to print to the template output

Example 4.3

```
<h1>Welcome, {{ username }}!</h1>
```

- The expression **{{ username }}** will be replaced with the value of the username variable provided by Flask when rendering the template. For example, if username = "John", then the result will be <h1>Welcome, John!</h1>.

3. **{#...#}** for comment not included in the template output.

Example 4.4

```
{# This is a comment, it won't appear in the rendered HTML #}
<p>This paragraph will be shown in the rendered HTML.</p>
```

- This comment will not be visible in the final HTML output. It is only used to provide explanations in the template that will not be displayed in the browser.

b) Template variables

Template variables are variables sent from Python (Flask backend) into HTML template files (Jinja2) via the render_template() function. These variables allow developers to display dynamic data on web pages, such as user names, product lists, form values, calculation results, or anything else that comes from the backend logic.

With template variables, HTML pages are no longer static, but can change according to conditions or user input, without having to create many separate HTML files. Variables are sent to HTML using syntax like this:

Syntax 4.2

```
render_template('template_name.html', vari_name1=value1, var_name2=value2, ..., var_name_N=value_N)
```

- **render_template()** is a Flask function to display the HTML file (template) to the browser.
- **template_name.html** is the name of the HTML file to be rendered, must be in the templates/ folder.
- **variable_name1=value1**, ... are data/variables from Python that are sent to the template and can be accessed in HTML using Jinja2 syntax: **{{ variable_name1 }}**, **{{ variable_name2 }}**, and so on.
- In **vari_name1**=**value1**, the variable on the left side belongs to the template (html). Therefore, the right side can be replaced with a variable that contains a value in the **".py"** file that renders the template.

Example 4.5

```
return render_template('profile.html', name='Haruka', age=21, city='Tokyo')
```

- **'profile.html'** is the name of the HTML template
- **name='Haruka', age=21, city='Tokyo'** are variables sent to the template and can be used with **{{ name }}, {{ age }}, {{ city }}** in the HTML.

The way to access variables in the template is:

1. Access attributes with dot notation (.)
   You can access the attributes of objects in templates using dot (.) notation, similar to Python:

Example 4.6

```
{{ foo.bar }}
```

2. Access attributes with bracket notation ([])

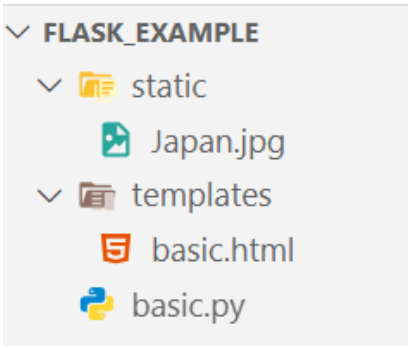
Example 4.7

```
{{ foo['bar'] }}
```

SIMPLE IMPLEMENTATION

a) Template Basic

1. Create a folder structure as follows



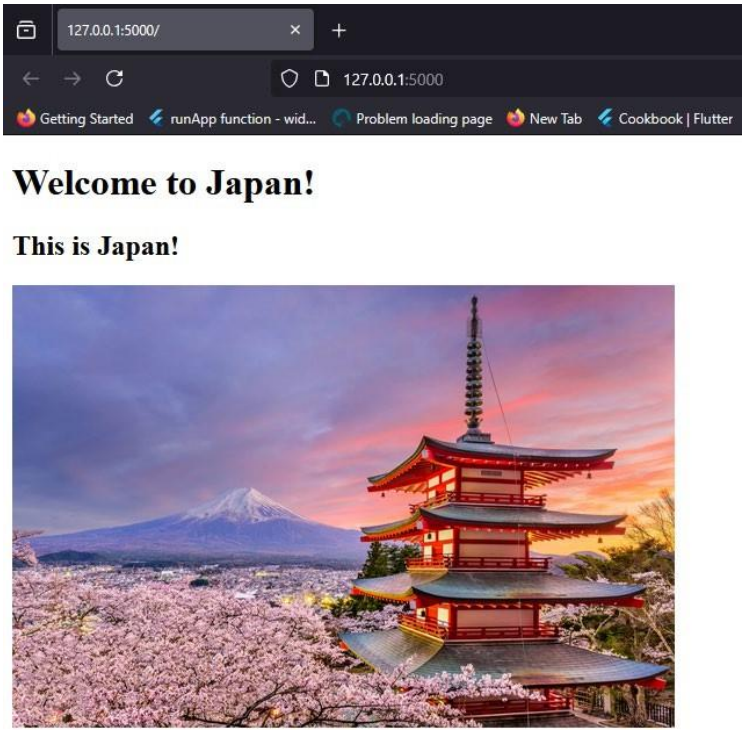
2. In the basic.html file, type this program code

```
<!DOCTYPE html>
<html lang=”en” dir=”ltr”>
    <head>
        <meta charset=”utf-8”>
        <title>
        </title>
    </head>
    <body>
        <h1>Welcome to Japan!</h1>
        <h2>This is Japan!</h2>
        <img src=”../static/Japan.jpg” alt=”Japan” style=”width: 30%;”>
    </body>
</html>
```

3. Type this program code in basic.py

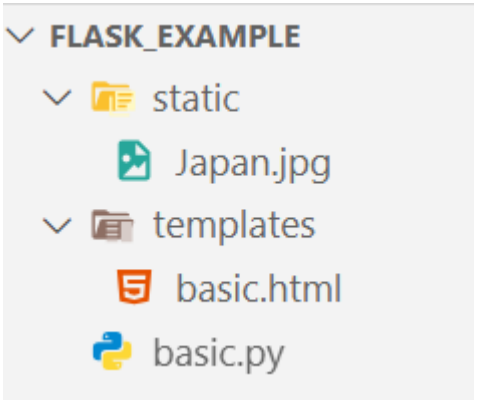
```
from  flask import Flask, render_template

app = Flask(__name__)

@app.route(‘/’)
def index():
    return render_template(‘basic.html’)

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

```

4. Run the program code, the output will appear as follows.



b) Template Variables

1. Create a folder structure as follows



2. In the "basic.html" file, type this program code

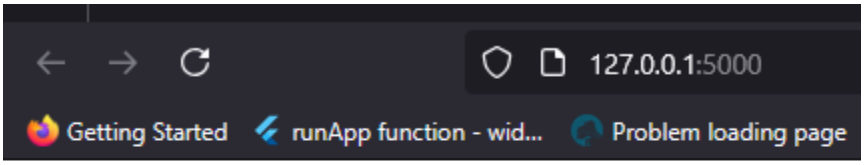
```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>
        </title>
    </head>
    <body>
        <h1>Hallo! {{name}}</h1>
        <h1>{{letters}}</h1>
    </body>
</html>
```

```
13
```

3. Type this program code in “basic.py”

```
1  from  flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route(‘/’)
6  def index():
7      name = “Haruka”
8      letters = list(name)
9
10     return render_template(‘basic.html’, name=name, letters=letters)
11
   if __name__==’__main__’:
       app.run(debug=True)
```

4. Run the program code, the output will appear as follows.

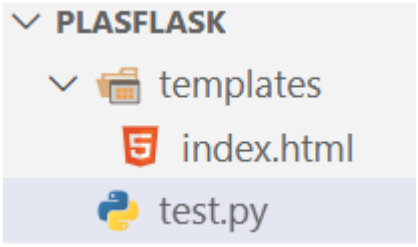


Hallo! Haruka

['H', 'a', 'r', 'u', 'k', 'a']

c) Template Variables with Dot Notation (.) or Bracket Notation ([])

1. Create a folder structure as follows



2. In the “index.html” file, type this program code

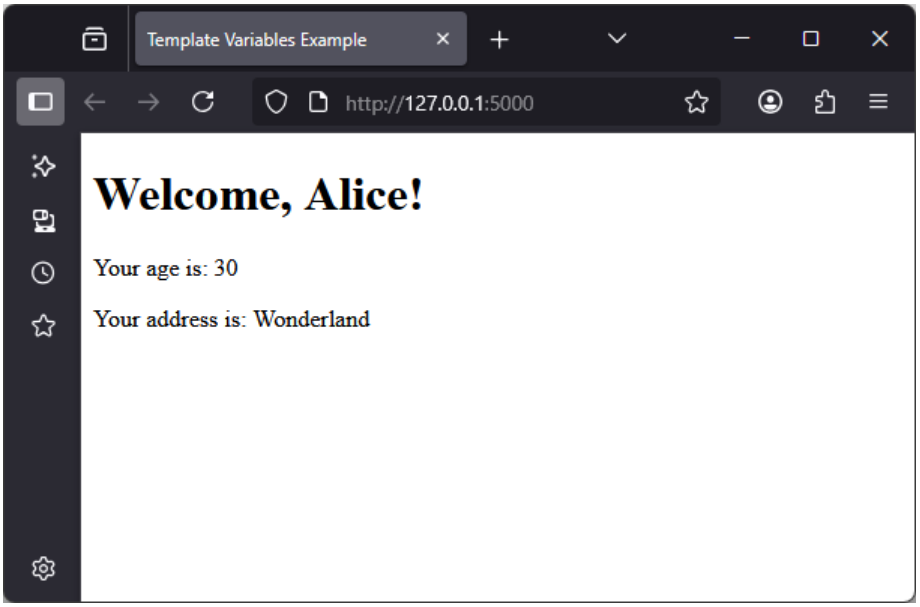
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Template Variables Example</title>
6  </head>
7  <body>
8      <h1>Welcome, {{ user.name }}!</h1>
9      <p>Your age is: {{ user.age }}</p>
10     <p>Your address is: {{ user['address'] }}</p>
```

```
11
12      <p>{{ user.email }}</p>
13  </body>
14  </html>
```

3. Type this program code in “test.py”

```
1   from flask import Flask, render_template
2
3   app = Flask(__name__)
4
5   @app.route('/')
6   def home():
7       user = {
8           'name': 'Alice',
9           'age': 30,
10          'address': 'Wonderland'
11      }
12      return render_template('index.html', user=user)
13
14  if __name__ == '__main__':
15      app.run(debug=True)
```

4. Run the program code, the output will appear as follows.



D. EXERCISE

	Exercise				
	VTP	GUP	BUP	CMP	EFP
Exercise Number Link	4.1, 4.2, 4.3, 4.4, 4.5	4.1, 4.2, 4.3	4.1	4.1, 4.2, 4.3	4.1