Python - Templates with Bottle Web App Framework (SimpleTemplate Engine)

Eli the Computer Guy - December 18, 2024
Version 1

Goal

This class will teach students how to use the SimpleTemplate Engine with Bottle Web App framework.

SimpleTemplate Engine

Template engines are a way Python web frameworks use to abstract the writing of dynamic web documents from the core Python code. It makes Bottle code easier to read because HTML files are separate files that are called instead of being written into the core code. They are generally more secure because the Template Engine will sanitize the variable values before they are printed to screen. So things such as HTML tags will be encoded so that they are shown but do not function.

Template Engines use Template Languages to display content. The language is slightly different between SimpleTemplate and Django. Generally Template Variables and Functions are wrapped in tags. There is both an opening tag, and closing one for things such as if/else statements and loops.

Setup

```
python3 –m pip install bottle
```

Create a views folder in your Current Working Directory

To find current working directory:
Go into the Python Shell and use the getcwd() in the os module to see the working directory.

```
python3
import os
print(os.getcwd())
exit
```

Then create a folder called **views** here.

Save your .tpl template files there.

Troubleshooting:
When saving a script as a .tpl VSCode may add a .h to the extension. You can get around this by saving the file while it is black as a .tpl, or by closing the file and renaming it after it has been created.

Basic

You can create a template within the main Bottle script. Simply call the template() function, type out the HTML and Template code, and then add the variable names to be used.

You create the variables in Python and then send the values with template_var_name=python_var_name.

This example uses a dynamic route to send a name to Bottle and then create an HTML page that says, "Hello <name>"

hello-name.py

```
from bottle import route, run, template

@route('/hello/<name>')
def index(name):
    return template('<h1>Hello {{name}}</h1>', name=name)

run(host='localhost', port='8080')
```

Static Template

Template files need to be stored in your working directory, generally your root. Or in a folder named **views** in your working directory.

When calling the template file you do not need to use the **.tpl** extension

hello-temp.py

```
from bottle import route, run, template

@route('/')
def index():
    return template('hello-template')

run(host='localhost', port='8080')
```

views/hello-template.tpl

```
<h1>Hello World</h1>

<p>This is a template</p>

<p>Templates are HTML files that you can add variables values to</p>
```

Include Templates

You can use include() to add other templates to your template. This is good for headers, footers, menus or any other elements that different pages should share.

include-temp.py

```
from bottle import run, route, template

@route('/')
def index():
    return template('template-include')

run(host='0.0.0.0', port='8080')
```

views/template-include.tpl

```
% include('header.tpl')

---- text on a page ---

% include('footer.tpl')
```

views/header.tpl

```
<h1>This is a Header</h1>
```

views/footer.tpl

```
<h2>This is a Footer</h2>
```

Variables and Templates

You can send variable values to a template and have the template dynamically add those values to the page. This is good for reusability and readability, and also has a security feature to sanitize variable values.

This example send a string and a list variable. You can iterate through the list using starting with a **%** for the for loop and ending it with **% end** .

In the following example try changing the name to a hyperlink.

var-temp.py

```
from bottle import route, run, template

@route('/')
def index():
    name = 'bob'
    list = ['item1', 'item2', 'item3', 'item4']
    return template('template-var', name=name, list=list)

run(host='localhost', port='8080')
```

views/template-var.tpl

```
<h1>Hello {{ name }}</h1>

<ul>
    % for x in list:
        <li>{{x}}</li>
    % end
</ul>
```

Dictionaries in Templates

You can iterate through a dictionary and access the keys and values using the items() function like you would in a standard Python script.

This example creates a list of student names with their ages and then prints them to an HTML table.

dict-temp.py

```
from bottle import route, run, template

@route('/')
def index():

    people = {
            'sue': 19,
            'tim': 33,
            'frank':20
    }

    return template('template-dict', people=people)

run(host='localhost', port='8080')
```

views/template-dict.tpl

```
<h1>Students</h1>

<table>
% for student, age in people.items():
    <tr><td>{{student}}</td><td>{{age}}</td></tr>
% end
</table>
```

If/ Else in the Template

You can run an if/else statement in the template. This can be good for if you want different aspects of a web page to be displayed based off of variable values.

You simply start with **%** before **if**, add **%** before **else** of **elif**, and then close with **% end** .

This example will simply check the age that is provided and then say whether the visitor is old enough to see the content.

if-temp.py

```
from bottle import route, run, template

@route('/')
def index():
    name = 'tim'
    age = 50
    return template('template-if', name=name, age=age)

run(host='localhost', port='8080')
```

views/template-if.tpl

```
<h1>Hello {{ name }}</h1>

% if age >= 21:
    <h2>You are old enough to see this</h2>
% else:
    <h2>You are too young</h2>
% end
```

Embed Python Code

You can embed Python code directly on the template. Generally this is not recommended. "Processing" should be done in the Python script for security and readability, but for complicated output it may be better to write the Python in the Template.

The tags for Python code are <% CODE %>. You can access variables by the names you created in the Python script, and any new variables created in the embedded code can be accessed in the template using {{ new_var }}.

python-temp.py

```
from bottle import route, run, template

@route('/')
def index():
    name = 'bob'
    payment = 20
    return template('template-python', name=name, payment=payment)

run(host='localhost', port='8080')
```

views/temp-python.tpl

```
<h1>Hello {{ name }}</h1>

<%
total = 100
time = total / payment
%>

<h2>It will take {{ time }} to pay off {{ total }} with payments of
{{ payment}}</h2>
```

CSS and Static Files

To add a CSS Style Sheet or any files such as images you will have to setup static files.

You will need to create a folder called **static** in your Working Directory, and then setup a static route function like you normally would.

In this example we've saved a CSS Style Sheet in the static folder, and then call that sheet from our template.

static-temp.py

```
from bottle import run, route, template, static_file

@route('/')
def index():
    name = 'bob'

    return template('template-static', name=name)

@route('/static/<filename:path>')
def send_static(filename):
    print(filename)
    return static_file(filename, root='./static/')

run(host='localhost', port='8080')
```

static/styles.css

```
h1 {
    background-color: red;
}

p {
    background-color: blue;
}
```

views/template-static.tpl

```
<link rel="stylesheet" href="/static/styles.css">

<h1>Hello {{ name }}</h1>

<p>This is text with CSS</p>
```