Flask Templates

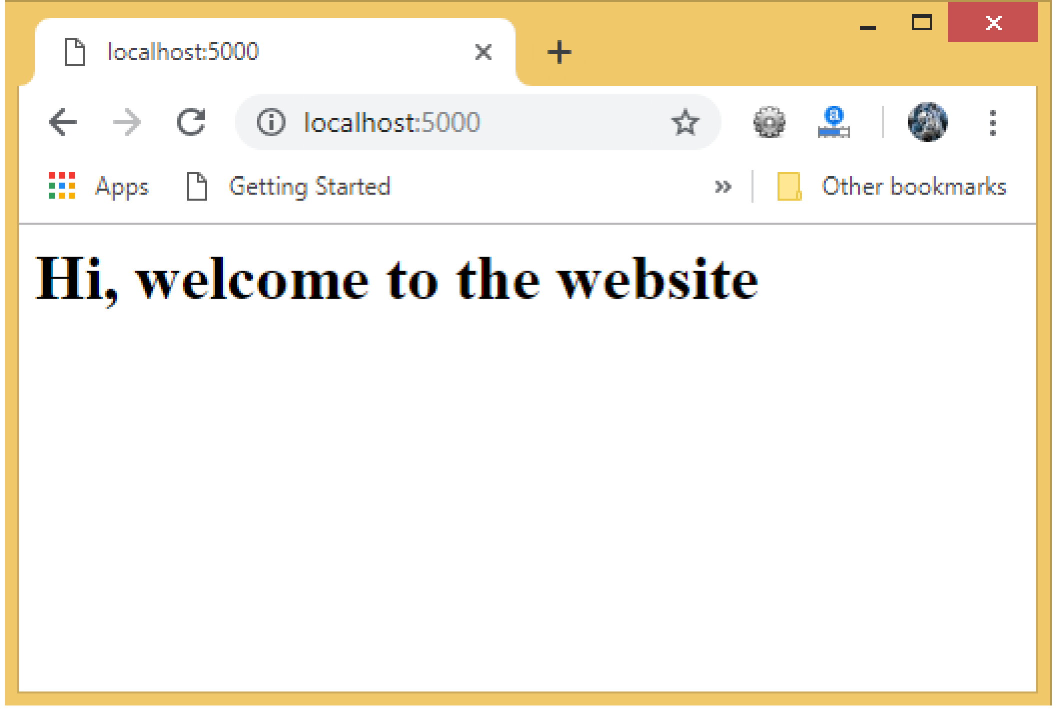
In the previous examples, we have returned the simple string as the response from the view function. Although, flask facilitates us to return the response in the form of HTML templates. In this section of the tutorial, we will go through the ways using which we can return the HTML response from the web applications.

Example

The following flask script contains a view function, i.e., the message() which is associated with the URL '/'. Instead of returning a simple plain string as a message, it returns a message with <h1> tag attached to it using HTML.

script.py

```
from flask import *
app = Flask(__name__)
@app.route('/')
def message():
      return "<html><body><h1>Hi, welcome to the website</h1></body></html>"
if __name__ == '__main__':
   app.run(debug = True)
```



Hi, welcome to the website

Rendering external HTML files

Flask facilitates us to render the external HTML file instead of hardcoding the HTML in the view function. Here, we can take advantage of the jinja2 template engine on which the flask is based.

Flask provides us the render_template() function which can be used to render the external HTML file to be returned as the response from the view function.

Consider the following example.

Example

To render an HTML file from the view function, let's first create an HTML file named as message.html.

message.html

```
<html>
<head>
<title>Message</title>
</head>
<body>
<h1>hi, welcome to the website </h1>
</body>
</html>
```

script.py

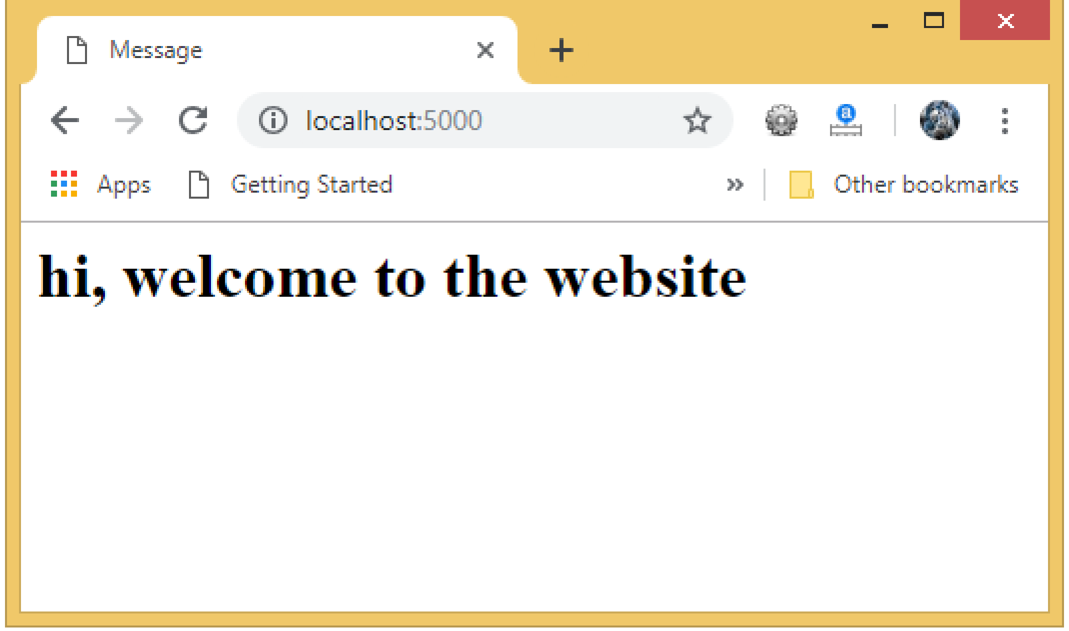
```
from flask import *
app = Flask(__name__)

@app.route('/')
def message():
      return render_template('message.html')
if __name__ == '__main__':
   app.run(debug = True)
```

Here, we must create the folder **templates** inside the application directory and save the HTML templates referenced in the flask script in that directory.

In our case, the path of our script file **script.py is E:\flask** whereas the path of the HTML template is **E:\flask\templates**.

Application Directory

- script.py
- templates
- message.html



hi, welcome to the website

Delimiters

Jinga 2 template engine provides some delimiters which can be used in the HTML to make it capable of dynamic data representation. The template system provides some HTML syntax which are placeholders for variables and expressions that are replaced by their actual values when the template is rendered.

The jinga2 template engine provides the following delimiters to escape from the HTML.

- {% ... %} for statements
- {{ ... }} for expressions to print to the template output
- {# ... #} for the comments that are not included in the template output
- # ... ## for line statements

Example

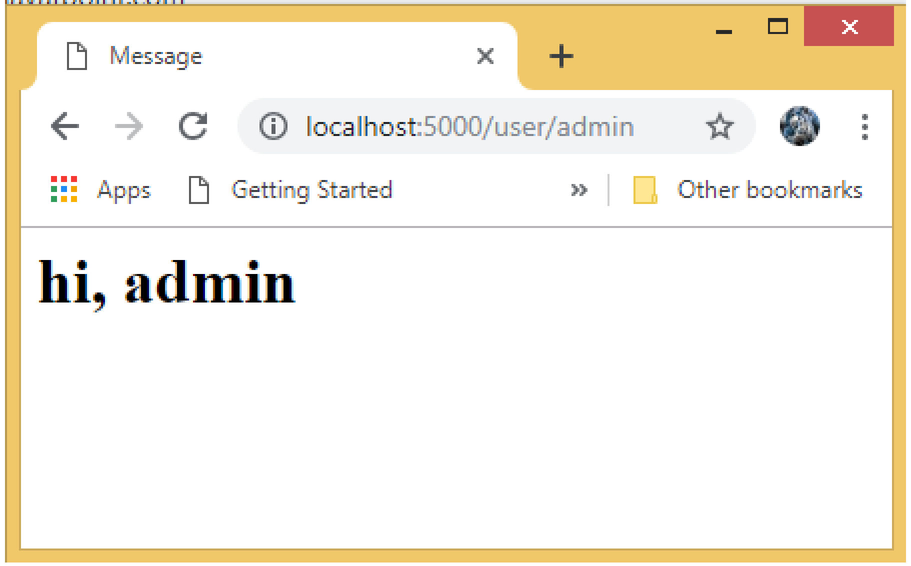
Consider the following example where the variable part of the URL is shown in the HTML script using the {{ ... }} delimiter.

message.html

```
<html>
<head>
<title>Message</title>
</head>
<body>
<h1>hi, {{ name }}</h1>
</body>
</html>
```

script.py

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

@app.route('/user/<uname>')
def message(uname):
      return render_template('message.html',name=uname)
if __name__ == '__main__':
   app.run(debug = True)
```



hi, admin

The variable part of the URL http://localhost:5000/user/admin is shown in the HTML script using the {{name}} placeholder.

Embedding Python statements in HTML

Due to the fact that HTML is a mark-up language and purely used for the designing purpose, sometimes, in the web applications, we may need to execute the statements for the general-purpose computations. For this purpose, Flask facilitates us the delimiter {%...%} which can be used to embed the simple python statements into the HTML.

Example

In the following example, we will print the table of a number specified in the URL, i.e., the URL http://localhost:5000/table/10 will print the table of 10 on the browser's window.

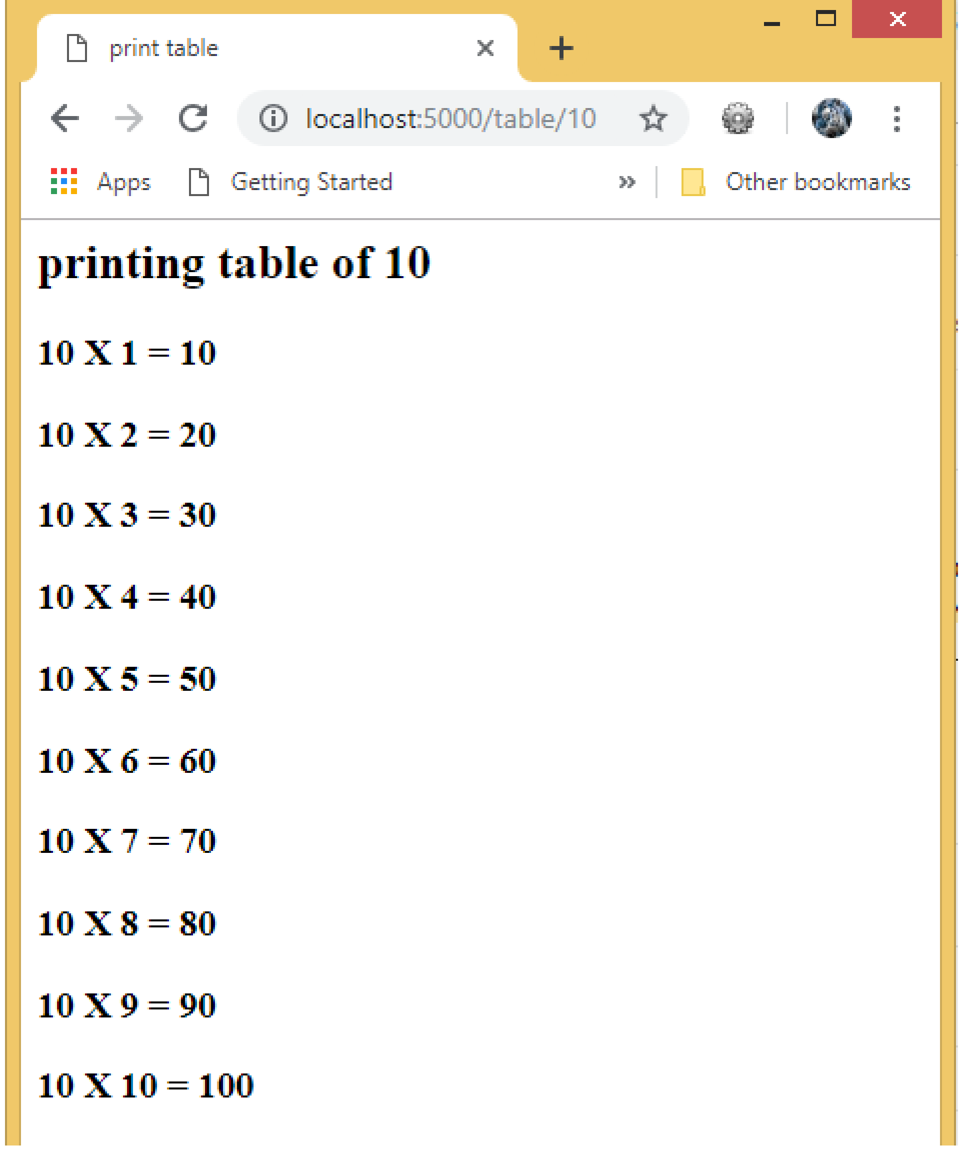
Here, we must notice that the for-loop statement is enclosed inside {%...%} delimiter, whereas, the loop variable and the number is enclosed inside {{ ... }} placeholders.

script.py

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

@app.route('/table/<int:num>')
def table(num):
      return render_template('print-table.html',n=num)
if __name__ == '__main__':
   app.run(debug = True)
```

print-table.py

```
<html>
<head>
<title>print table</title>
</head>
<body>
<h2> printing table of {{n}}</h2>
{% for i  in range(1,11): %}
    <h3>{{n}} X {{i}} = {{n * i}} </h3>
{% endfor %}
</body>
</html>
```



```
printing table of 10

10 X 1 = 10

10 X 2 = 20

10 X 3 = 30

10 X 4 = 40

10 X 5 = 50

10 X 6 = 60

10 X 7 = 70

10 X 8 = 80

10 X 9 = 90

10 X 10 = 100
```

Referring Static files in HTML

The static files such as CSS or JavaScript file enhance the display of an HTML web page. A web server is configured to serve such files from the static folder in the package or the next to the module. The static files are available at the path **/static** of the application.

Example

In the following example, the flask script returns the HTML file, i.e., **message.html** which is styled using the stylesheet **style.css**. The flask template system finds the static CSS file under **static/css** directory. Hence the style.css is saved at the specified path.

script.py

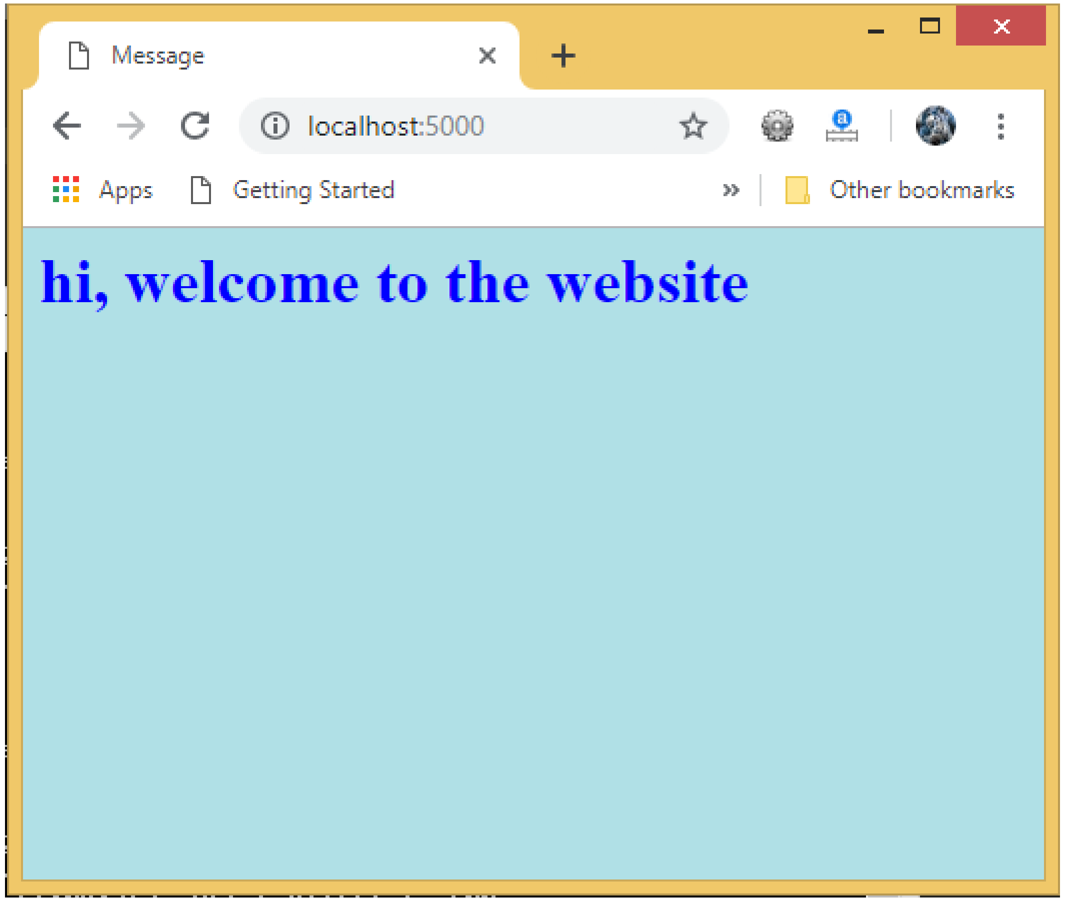
```
from flask import *
app = Flask(__name__)

@app.route('/')
def message():
      return render_template('message.html')
if __name__ == '__main__':
   app.run(debug = True)
```

message.html

```
<html>
<head>
    <title>Message</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>

<body>
    <h1>hi, welcome to the website</h1>
</body>
</html>
```

style.css

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```



hi, welcome to the website