Flask

Part 2

Hello World!

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def hello_world():

   return 'Hello World'


if __name__ == '__main__':

   app.run()
```

Flask constructor takes the name of current module (__name__) as argument.

The `route()` function of the Flask class is a decorator, which tells the application which URL should call the associated function.

Finally the `run()` method of Flask class runs the application on the local development server.

Optionally, we can debug the code by using an option, `app.run(debug = True)`

Hello World!

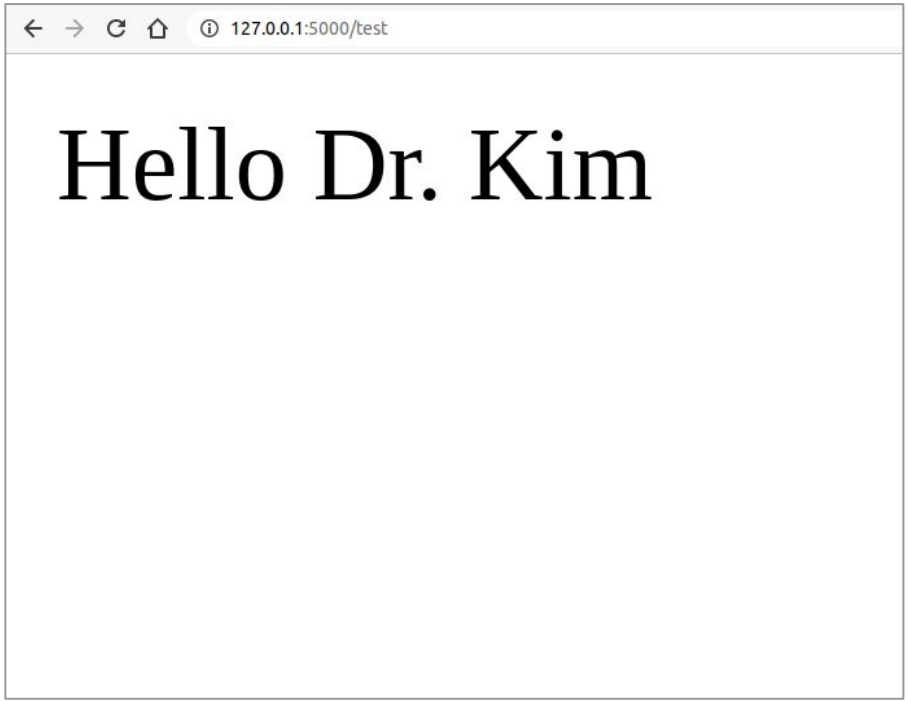
```
from flask import Flask

app = Flask(__name__)


def hello_world():

   return 'Hello World'

app.add_url_rule('/', 'hello_world', hello_world)


if __name__ == '__main__':

   app.run()
```

The `add_url_rule()` function of an application object is also available to bind a URL with a function as in the above example, `route()` is used.

```
/home/dkim/PycharmProjects/CSCI3328/venv/bin/python /home/dkim/PycharmProjects/CSCI3328/flask_hello.py
 * Serving Flask app "flask_hello" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```


127.0.0.1:5000
Apps
Mail - dongch…
My Drive - Goo…
Enrollment Se…
Nvidia Jetson…
Epoch - Real-ti…
The
Hello World

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

@app.route('/')
def hello_world():
   return 'Hello World'

@app.route('/test')
def hello_drkim():
   return 'Hello Dr. Kim'

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



Variable Rule

It is possible to build a URL dynamically, by adding variable parts to the rule parameter. This variable part is marked as <variable-name>. It is passed as a keyword argument to the function with which the rule is associated.

In the following example, the rule parameter of route() decorator contains <name> variable part attached to URL '/hello'. Hence, if the `http://localhost:5000/hello/Kim` is entered as a URL in the browser, `'Kim'` will be supplied to hello() function as argument.

```
from flask import Flask

app = Flask(__name__)


@app.route('/<name>')

def hello(name):

   return 'Hello %s!' % name


if __name__ == '__main__':

   app.run()
```

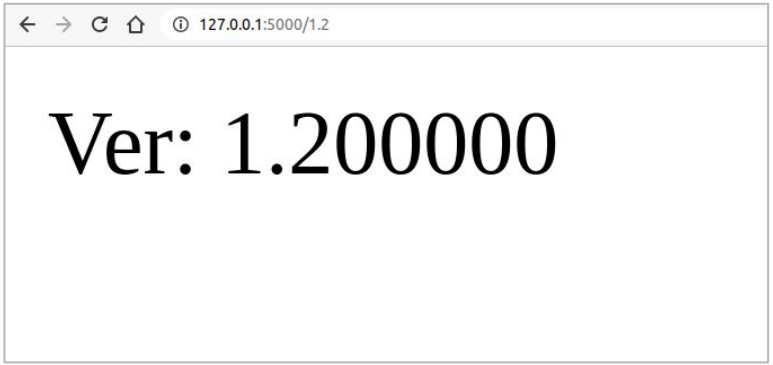

127.0.0.1:5000/Dr.%20Kim
Hello Dr. Kim!

Variable Rule - `int` and `float`

```
from flask import Flask

app = Flask(__name__)

@app.route('/<int:id>')

def article(id):

    return 'ID: %d' % id

@app.route('/<float:version>')

def code(version):

    return 'Ver: %f' % version
```

```
if __name__ == '__main__':

    app.run()
```


127.0.0.1:5000/365741
ID: 365741


127.0.0.1:5000/1.2
Ver: 1.200000

url_for()

The `url_for()` function is very useful for dynamically building a URL for a specific function.

The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.

```
from flask import Flask, redirect, url_for

app = Flask(__name__)


@app.route('/admin')

def hello_admin():

   return 'Hello Admin'


@app.route('/guest/<guest>')

def hello_guest(guest):

   return 'Hello %s as Guest' % guest


@app.route('/user/<name>')

def hello_user(name):

   if name =='admin':

      return redirect(url_for('hello_admin'))

   else:

      return redirect(url_for('hello_guest',guest = name))
```

url_for()


127.0.0.1:5000/admin
Hello Admin


127.0.0.1:5000/user/kim
127.0.0.1:5000/guest/kim
Hello kim as Guest

`http` methods - GET and POST

Http protocol is the foundation of data communication in world wide web.

By default, the Flask route responds to the GET requests. However, this preference can be altered by providing methods argument to route() decorator.

In order to demonstrate the use of POST method in URL routing, first let us create an HTML form and use the POST method to send form data to a URL.

login.html

```
<html>

   <body>

      <form action = "http://localhost:5000/login" method = " post">

         <p>Login</p>

         <p>ID <input type = "text" name = "id" /></p>

         <p>Password <input type = "text" name = "pass" /></p>

         <p><input type = "submit" value = "submit" /></p>

      </form>

   </body>

</html>
```

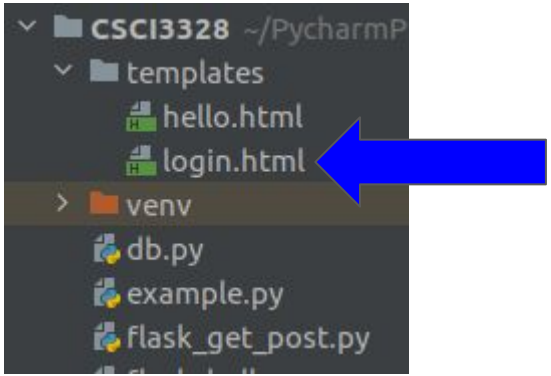
render_template()

```
from flask import Flask, redirect, url_for, request, render_template

app = Flask(__name__)


@app.route('/')

def static_file():

    return render_template('login.html')
```

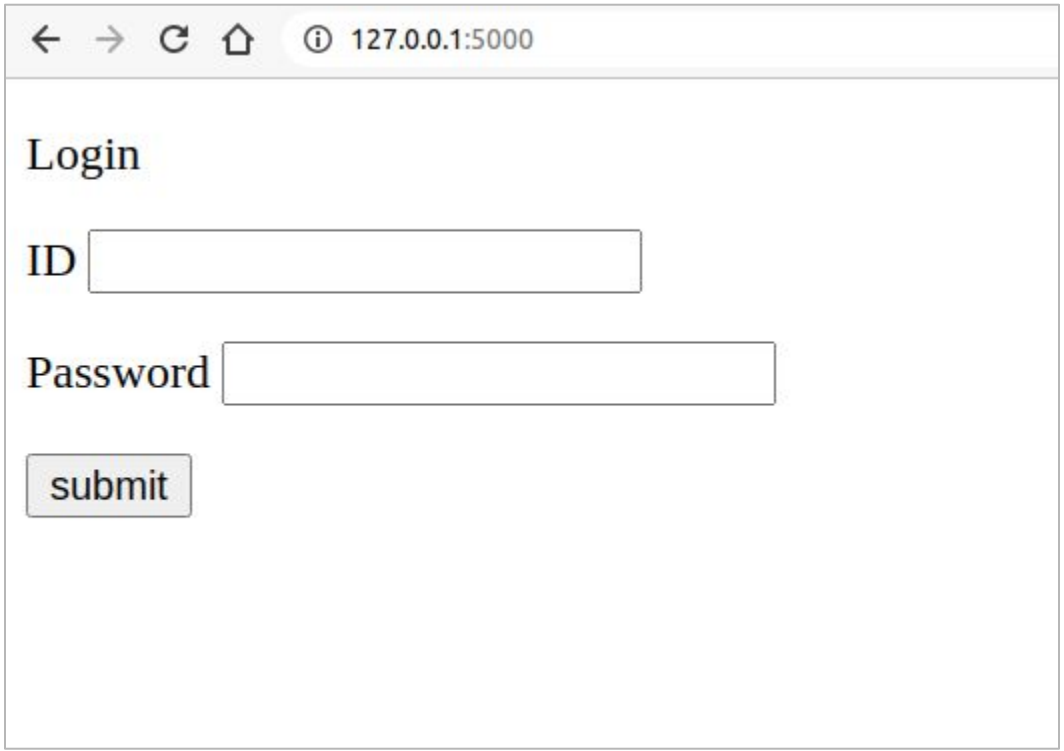
`render_template()`

Create a folder named "`templates`".

Put the login.html file in the folder.



login.html



```
from flask import Flask, redirect, url_for, request,
render_template

app = Flask(__name__)



@app.route('/')

def static_file():

    return render_template('login.html')



@app.route('/success/<id>')

def success(id):

   return 'welcome %s' % id
```

```
@app.route('/login',methods = ['POST', 'GET'])

def login():

   if request.method == 'POST':

      id = request.form['id']

      return redirect(url_for('success', id=id))

   else:

      id = request.args.get('id')

      return redirect(url_for('success', id=id))



if __name__ == '__main__':

   app.run(debug = True)
```

After the development server starts running, open `http://localhost:5000` in the browser, enter id and pass in the text fields and click Submit.

Form data is POSTed to the URL in action clause of form tag.

`http://localhost/login` is mapped to the `login()` function. Since the server has received data by POST method, value of `'id'` parameter obtained from the form data is obtained by −

```
id = request.form['id']
```


127.0.0.1:5000
Login
ID dkim
Password 1234
submit


localhost:5000/success/dkim
welcome dkim

Templates

It is possible to return the output of a function bound to a certain URL in the form of HTML. For example, `index()` function will render 'Hello World' with `<h1>` tag attached to it.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def index():

   return '<html><body><h1>Hello World</h1></body></html>'

if __name__ == '__main__':

   app.run(debug = True)
```

Templates

But it is too long.

Instead of returning hardcode HTML from the function, a HTML file can be rendered by the **`render_template()`** function.

For this, you have to create a folder **`templates`** and locate the html file under the folder.

```
from flask import Flask

app = Flask(__name__)



@app.route('/')

def index():

   return render_template('hello.html')



if __name__ == '__main__':

   app.run(debug = True)
```

Templates - passing a string

A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

```
<!doctype html>

<html>

   <body>

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

   </body>

</html>
```

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/hello/<user>')

def hello_name(user):

   return render_template('hello.html', name = user)

if __name__ == '__main__':

   app.run(debug = True)
```

Templates

The template engine uses the following delimiters for escaping from HTML.

{% ... %} for Statements

{{ ... }} for Expressions to print to the template output

{# ... #} for Comments not included in the template output

Templates - passing a dictionary

```
<html>
   <body>
      <table border = 1>
         {% for key, value in result.items() %}
            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
   </body>
</html>
```

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

@app.route('/result')
def result():
   dict = {'Python':100,'C++':90,'Java':80}
   return render_template('result.html',
result = dict)

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

Carmax + Flask

```python
from flask import Flask, render_template
import mysql.connector as mc

db = mc.connect(
    host="?",
    user="?",
    passwd="?",
    database="?"
)
print(db)
cursor = db.cursor()
cursor.execute("?")
rows = cursor.fetchall()

app = Flask(__name__)
@app.route('/lab')
def result():
    return render_template('inventory.html', rows=rows)


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

```html
<!doctype html>
<html>
    <body>
        <table border = 1>
            <thead>
                <td>VIN</td>
                <td>Brand</td>
                <td>Model</td>
                <td>Year</td>
            </thead>
            {% for row in rows %}
                <tr>
                    <td>{{row[0]}}</td>
                    <td>{{row[1]}}</td>
                    <td>{{row[2]}}</td>
                    <td>{{row[3]}}</td>
                </tr>
            {% endfor %}
        </table>

        <a href = "/">Go back to home page</a>
    </body>
</html>
```

Lab 25

Create a Flask application that connects to your car inventory database, either on Google Cloud SQL or Azure MySQL server, and displays the inventory in a web browser.