Introduction to Flask

Introduction à Flask (1/2)

- Lightweight WSGI Web Application Framework
  - Web Server Gateway Interface (WSGI) has been adopted as the standard for developing Python web applications.
  - WSGI is a specification for a universal interface between web server and web applications
- Built on Werkzeug (WSGI toolkit) and Jinja2 (web templating engine)
- Installation
  - pip install Flask
  - Conda install -c anaconda flask
- Developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco.

Introduction à Flask (2/2)

- It's a micro-framework
- Must use third-party libraries for database abstraction, form validation, upload handling, authentication
  - Integration with such tools is simple
- Used for back-end (server-side) development
- Used by:

Un premier exemple

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

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

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

- app = Flask(__name__) —an instance of the Flask class
- @app.route(‘/’) — to specify in which URL the hello_world() function will be called
- return ‘Hello, World!’— the message returned to the browser
- To run it:
  - On linux/Mac: export FLASK_APP=hello.py then flask run
  - On windows: set FLASK_APP=hello.py then flask run
  - Or python hello.py

Routing

- The syntax of the route() function : app.route(rule, options)
  - The rule parameter represents the URL binding with the function
  - Options are a list of parameters to pass to the underlying Rule object.
- The syntax of the run() function: app.run(host, port, debug, options) all its parameters are optional

Parameters	Description
host	Hostname to listen to. The default is 127.0.0.1 (localhost). Can be set to "0.0.0.0" to make the server available externally
port	By default it is 5000
debug	The default value is false. If set to true, debug information will be provided
options	To be passed to the underlying Werkzeug server.

Routing

- @app.route(str): specifies which URL will call the function
  - « / » is the main page
- @app.route(/path/<varname>): pass a variable (varname) as an argument to a function
  - Use “path:” to pass a variable containing a “/”
  - Optional: Optionally include the type —str:varname or int:varname

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

@app.route('/hello/<name>')
def hello_name(name):
   return 'Hello %s!' % name

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

http://localhost:5000/hello/SD

Hello SD!

Routing

- @app.route(str): specifies which URL will call the function
  - « / » is the main page
- @app.route(/path/<varname>): pass a variable (varname) as an argument to a function
  - Use “path:” to pass a variable containing a “/”
  - Optional: Optionally include the type —str:varname or int:varname

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

@app.route('/blog/<int:postID>')
def show_blog(postID):
   return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
   return 'Revision Number %f' % revNo

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

http://localhost:5000/blog/11

Blog Number 11

Routing

- Flask's URL rules are based on Werkzeug's routing module.
- This ensures that the URLs formed are unique and based on precedents set by Apache.

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

@app.route('/flask')
def hello_flask():
   return 'Hello Flask'

@app.route('/python/')
def hello_python():
   return 'Hello Python'

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

- Both rules look similar, but in the second rule, a trailing slash (/) is used. As a result, it becomes a canonical URL.
- Therefore, using /python or /python/ returns the same output. However, in the case of the first rule, /flask/ URL results in the 404 Not Found page.

Routing

- add_url_rule(): is a function that allows you to link a URL with a function
- The two codes below are equivalent:

```
@app.route('/hello')
def hello_world():
    return 'hello world'
```

```
def hello_world():
    return 'hello world'
app.add_url_rule('/', 'hello',hello_world)
```

Routing

- **url_for()**: accepts the name of a function as the first argument, and one or more keyword arguments, each corresponding to the variable part of the 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))
if __name__ == '__main__':
   app.run(debug = True)
```

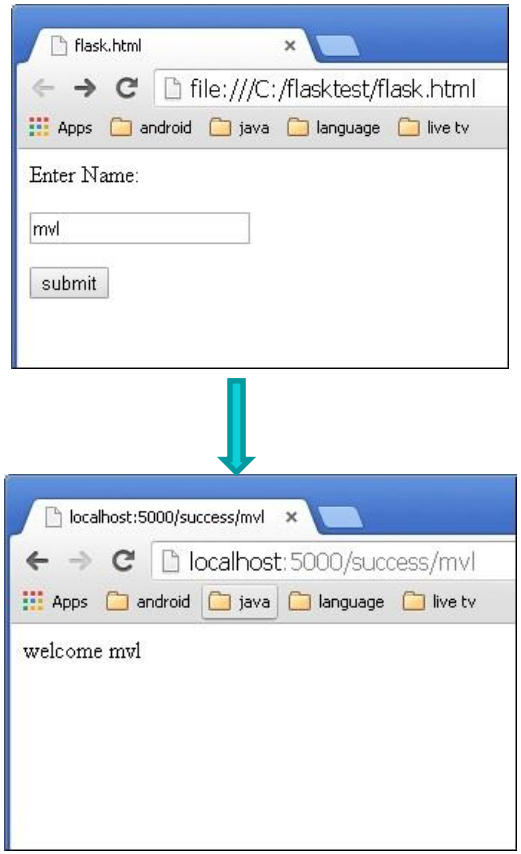
HTTP methods

- HTTP – Hypertext Transfer Protocol
- Used for client and server communication
  - The client sends a request and the server responds
- Flask creates a response from HTTP-compliant function returns
- HTTP methods refer to the type of requests received from clients

Méthode	Description
GET	Sends data in unencrypted form to the server. Most common method.
HEAD	Same as GET, but without response body
POST	Used to send HTML form data to the server. Data received by the POST method is not cached by the server.
PUT	Replaces all current representations of the target resource with the downloaded content.
DELETE	Removes all current representations of the target resource given by a URL

HTTP methods - example

```
<html>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Enter Name:</p>
         <p><input type = "text" name = "nm" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```




```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
   return 'welcome %s' % name

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST':
      user = request.form['nm']
      return redirect(url_for('success',name = user))
   else:
      user = request.args.get('nm')
      return redirect(url_for('success',name = user))

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

Templates

- It is possible to return the output of a function linked to a certain URL as HTML
- In the following script, the hello() function will render "Hello World" with the <h1> tag attached.

```
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)
```

- Generating HTML content from Python code is tedious, especially when variable data and Python language elements such as conditionals or loops need to be placed.

Templates: the Jinja2 template engine

- This is where we can take advantage of the Jinja2 templating engine, on which Flask is based.
- Instead of returning HTML from the function, an HTML file can be rendered by the render_template() function.

- Flask will try to find the HTML file in the "templates" folder, in the same folder where this script is present.

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

@app.route('/')
def index():
   return render_template('hello.html')

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

- Application folder
  - Hello.py
  - templates
    - hello.html

The term "Web templating system" refers to the design of an HTML script into which variable data can be dynamically inserted.

Les templates: le moteur de template Jinja2

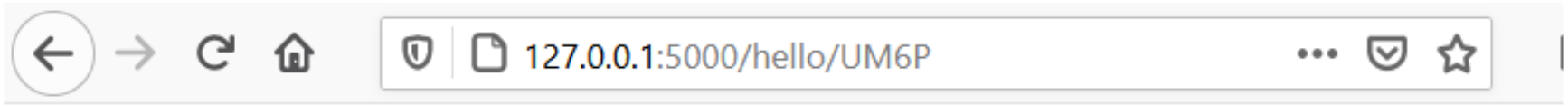
Exemple

hello.html

```
<!doctype html>
<html>
  <body>

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

  </body>
</html>
```

main.py

```
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)
```



Hello UM6P!

Templates: the Jinja2 template engine

- The jinja2 templating engine uses the following delimiters to escape HTML.
  - {% ... %} for declarations
  - {{ ... }} for expressions to print on the template output
  - {# ... #} for comments not included in the template output
  - # ... ## for one-line declarations

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

@app.route('/hello/<int:score>')
def hello_name(score):
   return render_template('hello.html',
marks = score)

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

```
<!doctype html>
<html>
   <body>
      {% if marks>50 %}
         <h1> Your result is pass!</h1>
      {% else %}
         <h1>Your result is fail</h1>
      {% endif %}
   </body>
</html>
```

Templates: the request object

- Data from a client's web page is sent to the server as a global request object.
- In order to process the query data, it must be imported from the Flask module.
- Important attributes of the query object are listed below

Attribut	Description
Form	Il s'agit d'un objet dictionnaire contenant des paires clé et valeur de paramètres de formulaire et leurs valeurs.
args	contenu analysé de la chaîne de requête qui fait partie de l'URL après le point d'interrogation (?).
cookies	objet dictionnaire contenant les noms et les valeurs des cookies.
files	les données relatives au fichier téléchargé.
method	méthode de demande actuelle

Les templates: l'objet request

Exemple: l'envoie des données d'un formulaire vers un template

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

@app.route('/')
def student():
   return render_template('student.html')

@app.route('/result',methods = ['POST', 'GET'])
def result():
   if request.method == 'POST':
      result = request.form
      return render_template("result.html",result = result)

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

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

result.html

action="{{ url_for('result')}}"

```
<html>
   <body>
      <form action = "http://localhost:5000/result" method = "POST">
         <p>Name <input type = "text" name = "Name" /></p>
         <p>Physics <input type = "text" name = "Physics" /></p>
         <p>Chemistry <input type = "text" name = "chemistry" /></p>
         <p>Maths <input type ="text" name = "Mathematics" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

Student.html