Programmation Serveur avec Flask

Objectf

Écrire un programme qui est un serveur Web.

Outils

Utiliser Python et un module qui implémente déjà HTTP

- django
- bottle
- **flask**
- cherrypi
- ...

Flask

- Simplicité $\approx$ bottle, mais avec quelques fonctionnalités (débuggage, modules supplémentaires...) intéressantes.
- Loin derrière Django niveau fonctionnalités

Principe

- URL ➡️ fonction
- Consultattion de l'URL ➡️ exécution de la fonction ➡️ retour de la fonction = contenu de la page

Exemple

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


@app.route('/')
def hello():
    return "Hello World!"

@app.route('/bye')
def bye():
    return "Have a nice day"


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

test 1

test 2

Accès aux paramètres GET et / ou POST

```
@app.route('/ajoute')
def ajoute():
    a = int(flask.request.args.get('a', '0'))
    b = int(flask.request.args.get('b', '0'))
    return str(a + b)
```

⚠️ Les paramètres sont des `str` ainsi que le retour de la fonction

Voir aussi à `request.forms` pour les requêtes POST`

test

URL variable

```
@app.route('/mul/<int:a>/<int:b>')
def mul(a, b):
    return str(a * b)
```

test

Utilisation des templates

```
@app.route('/exp/<int:a>/<int:b>')
def exp(a, b):
    return flask.render_template("operation.html",
            a=a, b=b,
            operation="**",
            resultat=a**b)
```

```
{% extends "default.html" %}
{% block main %}
{{a}} {{operation}} {{b}} donne <span class="label">{{resultat}}
{% endblock %}
{% block footer %}
{{ super() }} - et avec soin
{% endblock %}
```

test

```
<!doctype html>
<html lang="fr">
<head>
{% block head %}
  <meta charset="utf-8">
  <meta name="viewport"
        content="width=device-width, initial-scale=1">
  <title>{{title}}</title>
  <link rel="stylesheet"
        href="https://unpkg.com/picnic@6.5.0/picnic.min.css">
  <style>main, footer {max-width:600px; margin:auto}</style>
  {% endblock %}
</head>
<body>
  <main>
   {% block main %} Contenu par défaut.  {% endblock %}
   </main>
<hr>
<footer>
{% block footer %} Page réalisée avec Flask {% endblock %}
</footer>
</body>
</html>
```

Renvoyer du JSON

```
@app.route('/pgcd/<int:a>/<int:b>')
def pgcd(a, b):
    gcd = math.gcd(a, b)
    return flask.jsonify({"op1": a, "op2": b,
                          "ope": "pgcd", "res": gcd})
```

test

Servir une image

```
import math
import matplotlib.pyplot as plt
import flask
import io

app = flask.Flask(__name__)
@app.route('/')
def trace():
    x = [0.01 * _ for _ in range(614)]
    y = [math.sin(_) for _ in x]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(x, y)
    imgdata = io.BytesIO()
    fig.savefig(imgdata, format='png')
    imgdata.seek(0)
    resp = flask.Response(imgdata.read())
    resp.headers['Content-type'] = 'image/png'
    return resp
if __name__ == '__main__':
    app.config['DEBUG'] = True
    app.run()
```

Manipulation d'images (ou autres objets binaires)

```
import requests
from PIL import Image
import base64
import io

# il faut le numéro de séquence exact (avec les 0)
url = "https://oeis.org/A000045/graph?png=1"
r = requests.get(url)

# PIL
img = Image.open(io.BytesIO(r.content))
img.show()

# Inline image
b64 = base64.b64encode(r.content)
open("test.html","w+").write('<html><body>\
  <img src="{0}"/></body></html>'.\
  format(b64.decode("ascii")))
```

Autres fonctionnalités utiles

- `flask.session['KEY']` : utilisation de variables de session
- `flask.abort(403, "message")`
- `flask.url_for(function)` : url qui provoque l'appel de la fonction donnée
- `flask.redirect(...)` :
- pages statiques dans le rép `static` servies automatiquement

FIN