TP1 Flask

Qu'est-ce qu'un *framework* ?

- Une boîte à outils logicielle
- destinés à *un certain type* d'applications Web,
- offrant des fonctionnalités de haut niveau,
- et favorisant la mise en œuvre de bonnes pratiques.

Frameworks Web en Python

- Django
- Flask
- Pyramid
- CherryPy
- Twisted
- ...

Premiers pas en Flask

Hello World Flask

```
from flask import Flask

app = Flask(__name__)

@app.route("/greeting")
def greeting():
    return "Hello world"
```

Explications

- `app` est une *application* Flask.
- La fonction `greeting` est appelée une *vue*. Elle retourne une chaîne de *caractères*, qui sera le contenu de la réponse, de type HTML.
- La ligne qui précède la fonction `greeting` est un **décorateur** python. Il sert à indiquer l'URL pour laquelle cette vue doit être utilisée.

Serveur de développement

- Flask fournit son propre *serveur de développement*, directement dans la méthode `run` de l'application.
  On peut donc créer un script `serveur.py` contenant simplement :

  ```
  # en supposant que votre application Flask est définie
  # dans un fichier nommé 'mon_projet.py'
  from mon_projet import app
  app.run(debug=True)
  ```

- Le mode `debug` offre de plus des fonctions avancées, notamment :
  - le recharchement automatique des fichiers python en cas de modification,
  - l'affichage des exceptions dans le navigateur,
  - la possibilité d'interagir avec le code python depuis le navigateur en cas d'erreur.

Routes

- En développement Web, on appelle route une URL ou un ensemble d'URLs conduisant à l'exécution d'une fonction donnée.
- Dans Flask, les routes sont déclarées *via* le décorateur `app.route`, comme dans l'exemple ci-dessus.

- Une route peut être *paramétrée*, auquel cas le paramètre sera passé à la fonction vue :

```
@app.route("/hello/<name>")
def hello(name):
    return "Hello %s" % name
```

Réponse personnalisée

Il est possible pour une vue de retourner un objet `Response` (au lieu d'une chaîne de caractères) dont on peut alors personnaliser les méta-données. Cet objet peut être produit grâce à la fonction `flask.make_response`.

Exemple :

```
@app.route("/some.pdf")
def some_pdf():
    pdf_data = produce_pdf_data()
    resp = make_response(pdf_data)
    resp.headers["content-type"] = "application/pdf"
    return resp
```

Génération d'URL

- Les routes permettent à Flask de trouver la vue correspondant à une URL, mais également de faire **l'inverse**, à savoir de reconstruire l'URL d'une vue donnée.
- La fonction `flask.url_for` prend en paramètre le nom d'une vue (le nom de la fonction, dans une chaîne de caractères), avec ses éventuels paramètres, et retourne l'URL correspondante. Exemples :

```
# avec les routes des exemples précédents
url_for('root')               # → "/"
url_for('hello', name="John")  # → "/hello/John"
```

- on peut passer à `url_for` des paramètres supplémentaires (*i.e.* non spécifiés par la vue), lesquels seront ajoutés en paramètres d'URL :

```
url_for('hello', name="John", foo="bar")  # → /hello/John?foo=bar
```

- le paramètre `_external` peut être mis à *True* pour généner une URL absolu :

```
url_for('root', _external=True)  # → http://localhost:5000/
```

Exemple d'utilisation :

```
@app.route("/about")
def about():
    return """<a href="%s">Retour à la page d'accueil</a>""" % \
        url_for('root')
```

Ressources statiques

- répertoire `static`, situé dans le même répertoire que le fichier Python définissant l'application.
- L'URL de ces fichiers est donnée par la fonction `url_for` :

```
url_for("static", filename="nom_du_fichier.css")
```