flask

May 6, 2015

```
In [5]: %%javascript
        IPython.Cell.options_default.cm_config.lineNumbers = false;
```

```
<IPython.core.display.Javascript at 0x7fa0a8036f98>
```

```
In [12]: # Charge ma feuille de style pour nbviewer
         from IPython import utils
         from IPython.core.display import HTML
         import os
         def css_styling():
             base = utils.path.get_ipython_dir()
             styles = "<style>\n%s\n</style>" % (open('./documents/custom.css','r').read())
             return HTML(styles)
         css_styling()
```

```
Out[12]: <IPython.core.display.HTML at 0x7fa0a8044e10>
```

1 IV Site internet en python avec flask

Classiquement sur internet on utilise un serveur ***lamp*** :
linux (os), **a**pache (serveur), **m**ysql (base de données), **p**hp (language de programmation pour avoir des pages dynamiques).

Il existe de nombreux outils pour le web écrit en python: *serveur Web* (Zope, gunicorn) ; *script cgi*; *frameworks Web* (Flask, Django, cherrypy etc . . . ,

nous utiliserons le framework ***flask***. Pour se documenter je vous conseille deux sites.
- http://flask.pocoo.org/docs - http://openclassrooms.com/courses/creez-vos-applications-web-avec-flask - http://pub.phyks.me/sdz/sdz/creez-vos-applications-web-avec-flask.html

Nous avons besoin de quelques fichiers pour commencer.

- Télécharger le fichier https://github.com/debimax/cours-debimax/raw/master/documents/isn-flask.tar.gz.
- Décompressez le (en console : tar zxvf isn-flask.tar.gz).

1.1 1) Flask et les templates

1.1.1 a) Le principe

Avec *Flask* on pourrait mettre le code dans un seul *fichier.py* mais on utilise de préférence des *templates* (avec *jinja2*).

L'arborescence d'un projet Flask sera :

```
projet/script.py
projet/static/
projet/templates
projet/templates/les_templates.html
```

Les templates se mettent dans le dossier ***templates/***. Le dossier ***static/*** contiendra lui toutes les images, fichiers css, js ...

Ouvrez avec *geany* le fichier ***exemple.py*** :

```
01 #!/usr/bin/python3
02 # -*- coding: utf-8 -*-
03 from flask import Flask, render_template, url_for
04 app = Flask(__name__)   # Initialise l'application Flask
05
06 @app.route('/')  # C'est un décorateur, on donne la route ici "/"  l'adresse sera donc localhost:500
07 def accueil():
08     lignes=['ligne {}'.format(i) for i in range(1,10)] # Que fait cette ligne?
09     return render_template("accueil.html", titre="Bienvenue !",lignes=lignes) # On utilise le templa
10
11 if __name__ == __main__ :
12     app.run(debug=True)
```

Dans le code ci-dessous, quel est le contenu de la variable ***lignes***?

```
liges=['ligne{}'.format(i) for i in range(1,10)]
```

.....

Ouvrez maintenant le fichier ***templates/accueil.html***.

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04 <meta charset="utf-8" />
05 <title>{{ titre }}</title>
06 </head>
07 <body>
08 <header><h1>{{ titre }}</h1></header>  <!-- On affiche le titre -->
09 <section>
10 <ul>
11 {% for ligne in lignes %}   <!-- Il existe avec jinja des boucles des conditions ... -->
12 <li>{{ ligne }}</li>        <!-- On affiche le contenu de la liste ligne -->
13 {% endfor %}
14 </ul>
15 </section>
16 </body>
17 </html>
```

Exécuter le fichier ***exemple.py*** depuis *geany*. Ouvrez un navigateur internet à l'adresse `http://localhost:5000` (celui qui n'est pas écrit pablo car il ne doit pas utiliser de proxy).

Le fichier *templates/accueil.html* s'appelle un ***template*** et le logiciel qui gère ces templates est ***jinja*** (on dit aussi moteur de templates).

1.1.2 b) Ajouter la date

Il faudra importer la librairie ***time*** puis utiliser le décorateur ***@app.context_processor*** pour passer l'heure à tous les templates.

```
03 from flask import Flask, render_template, url_for
04 import time           # pour afficher l'heure
05 app = Flask(__name__) # Initialise l'application Flask
06
07 @app.context_processor   # ceci pour envoyer à tous les templates les variables date et heure
08 def passer_date_heure():
09     date=time.strftime('%d/%m/%Y',time.localtime())
10     heure=time.strftime("%H:%M:%S",time.localtime())
11     return dict(date=date,heure=heure)
12
13 @app.route('/')
```

Il faut maintenant modifier le template ***accueil.html***

```
15     </section>
16    <footer>Nous sommes le {{ date }} et il est {{ heure }} heures.</footer>
17     </body>
```

Il suffit d'enregistrer le fichier ***exemple.py*** s'il n'y a pas d'erreur pour que le site soit mis à jour automatiquement.
Observez le résulat puis rechargez plusieurs fois la page.

Question: Pourquoi est il peu judicieux d'utiliser le serveur python pour afficher l'heure?

.....
.....

1.1.3 c) Les fichiers css

Nous allons utiliser un fichiers *css* pour l'habillage (présentation) de notre page web.

Modifions le fichier ***templates/accueil.html***.

```
05 <title>{{ titre }}</title>
06 <link href="{{ url_for('static',filename='mon_style.css')}}" rel="stylesheet" type="text/css" />
07 </head>

17    <footer>Nous somme le <em class="Rouge">{{ date }}</em> et il est <em class="Rouge">{{ heure }}</e
```

Dans le fichier ***static/mon_style.css*** il y a ceci:

```
01 html, body {
02    height: 100%;
03    margin: 0;
04    padding: 0;}
05
06 body {
07 display:table;
08 width:100%;}
09
10 header,section,footer {display:table-row;}
11
12 section {height:100%;}
13
14 h1 {
15 color: #ff00ff;
16 text-align: center;19 }
17
18 footer {
```

```
19 width: 100%;
20 height: 20px;
21 background-color: #f5f5f5;
22 text-align: center;}
23
24 em.Rouge {color: #ff0000;}
```

- Je met le contenu de la balise
  <h1>
  au centre et en couleur.

- Je place le 'footer' au centre en bas de la page avec un fond de couleur.

- Je met en rouge avec la classe ***em.Rouge***.
- Le reste c'est pour placer les sections et avoir un footer au bas de la page.

1.1.4 d) Le javascript

Coté client (le navigateur) on utilise aussi des scripts dans le code HTML des pages web comme javascript, jquery etc...

Nous allons maintenant utiliser un fichier ***jascript (.js)*** pour afficher l'heure. Le code javascript sera effectué par le navigateur, ce n'est donc pas le serveur qui exécute le code et donc pas besoin de rafraichir la page internet pour que le code soit exécuté.
Il n'est pas question d'expliquer le code javascript mais vous pouvez regarder à quoi cela ressemble dans le ficher ***static/mes_scripts.js***.

Comme pour le fichier *.css* Il faut indiquer le fichier *.js* aux templates.

Modifions le fichier ***templates/accueil.html***.

```
 6 <link href=" url_for('static', filename='mon_style.css') " rel="stylesheet" type="text/css" />
 7 <script type=text/javascript src="{{url_for('static', filename='mes_scripts.js') }}"></script>
 8 </head>

17 </ul>
18 <footer>Nous sommes le <em class=Rouge id="date"></em> <script type="text/javascript"> window.onload
19 </body>
```

Si cela fonctionne alors on peut **enlever** dans ***accueuil.py*** la fonction ***passer_heure()*** pour obtenir

```
03 from flask import Flask, render_template, url_for
04 app = Flask(__name__) # Initialise l'application Flask
05
06 @app.route('/')
```

1.1.5 e) Les images

Ajoutons un favicon. Vous savez c'est la petite images à gauche de l'adresse dans un navigateur. Le nom de l'image doit être ***favicon.ico***.

J'ai déjà mis deux images dans le dossier ***static/***.
Il suffit donc de rajouter dans ***templates/accueil.html***

```
08 <link rel="shortcut icon" href="{{ url_for('static' , filename='favicon.ico')}}">
09 </head>
10 <body>
11 <header><h1>{{ titre }} <img src="{{ url_for('static',filename ='fleur.png') }}" alt="fleur" title="f
12 <section>
```

1.2 2) Les formulaires

Dans le protocole ***HTTP***, une ***méthode*** est une Commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action. En général l'action concerne une ressource identifiée par l'URL qui suit le nom de la méthode

- La méthode ***get***:
  C'est la méthode la plus courante pour demander une ressource.
- La méthode ***post***:
  Cette méthode est utilisée pour transmettre des données en vue d'un traitement à une ressource (le plus souvent depuis un formulaire HTML). L'URI fournie est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

1.2.1 a) Exemple de formulaire utilisant la méthode POST

Une page internet utilise souvent des champs de formulaire avec la balise ***INPUT*** et il existe de nombreux champs de formulaires. Regardez sur ce site quelques possibilités http://www.startyourdev.com/html/tag-html-balise-input

Modifions la page ***templates/accueil.html*** pour mettre un *formulaire*

```
17  </ul>
18  <div id="content">
19     <form method="post" action="{{ url_for('hello') }}">
20     <label for="nom">Entrez votre nom:</label>
21     <input type="text" name="nom" /><br />
22     <label for="prenom">Entrez votre prénom:</label>
23     <input type="text" name="prenom" /><br />
24     <input type="submit" />
25     </form>
26 </div>
27 </section>
```

pour le fichier ***exemple.py*** on importe ***flask.request***

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

et on crée une nouvelle route.

```
09      return render_template("accueil.html", titre="Bienvenue !",lignes=lignes)
10
11 @app.route('/hello/', methods=['POST'])
12 def hello():
13     nom=request.form['nom']
14     prenom=request.form['prenom']
15     return render_template('page2.html' ,titre="Page 2", nom=nom, prenom=prenom)
```

Il ne reste plus qu'à créer le deuxième template ***templates/page2.html*** *(pensez à copier les lignes depuis templates/accueil.html).*

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04 <meta charset="utf-8" />
05 <title>{{ titre }}</title>
06 <link href="{{ url_for('static', filename='mon_style.css') }}" rel="stylesheet" type="text/css" />
```

```
07 <script type="text/javascript" src="{{url_for('static', filename='mes_scripts.js') }}"></script>
08 <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
09 </head>
10 <body>
11 <h1>{{ titre }} <img src="{{ url_for('static', filename = 'fleur.png') }}"  alt="fleur" title="fleur"
12 <section>
13 <p>Bonjour <b>{{ prenom }} {{ nom }}.</b></p>
14 </section>
15 <footer>Nous sommes le <em class="Rouge" id="date"></em><script type="text/javascript">window.onload=
16 </body>
17 </html>
```

1.2.2 b) Formulaire qui redirige vers la même page

Il est nécessaire d'utiliser à la fois la méthode ***POST*** et la méthode ***GET***.
La méthode ***GET*** quand on arrive sur la page, puis la méthode ***POST*** quand on transmet les informations.

Modifier le fichier *exemple.py*

```
03 from flask import Flask, render_template, url_for, request
04 app = Flask(__name__) # Initialise l'application Flask
05
06 @app.route('/', methods=['GET','POST'])  # On doit indiquer que l'on utilise les deux méthodes
07 def accueil():
08 lignes=['ligne {}'.format(i) for i in range(1,10)]
09 if request.method == 'GET':   # à la 1° connection c'est uné méthode GET
10     nom=''                    # La variable doit être définie
11     prenom=''
12     titre="Bienvenue !"
13 else:                         # à la 2° connection c'est une méthode POST
14     nom=request.form['nom']
15     prenom=request.form['prenom']
16     titre="Méthode POST"
17 return render_template("accueil.html", titre=titre,lignes=lignes,nom=nom,prenom=prenom)
18
19 if __name__ == '__main__':
20 app.run(debug=True)
```

et le template acceuil.html

```
17 </ul>
18
19 {% if nom == '' and prenom == '' %}
20 <div id="content">
21      <form method="post" action="{{ url_for('accueil') }}">
22      <label for="nom">Entrez votre nom:</label>
23      <input type="text" name="nom" /><br />
24      <label for="prenom">Entrez votre prénom:</label>
25      <input type="text" name="prenom" /><br />
26      <input type="submit" />
27      </form>
28 </div>
29 {% else %}
30 <p>bonjour {{ prenom }} {{ nom }}</p>
31 {% endif %}
32 </section>
```

1.3 3) Une partie commune à toutes les pages

Nous avons plusieurs templates qui utilisent le même bas de page.
Il est donc possible d'utiliser un seul fichier template et de l'inclure dans les autres. Créer un fichier ***templates/footer.html***

```
<footer>Nous sommes le <em class="Rouge" id="date"></em> <script type="text/javascript">window.onload =
```

Puis modifier les templates ***templates/accueil.html***

```
27 </div>
28 {% include 'footer.html' %}
29 </body>
```

et ***templates/page2.html***.

```
14 <p>Bonjour <b>{{ prenom }} {{ nom }}.</b></p>
15 {% include 'footer.html' %}
16 </body>
```

1.4 4) Mettre le site sur internet

Habituellement on s'héberge soit même mais il est possible de mettre son site chez un hébergeur. Il en existe plusieurs qui autorisent les scripts python.

Ouvrez le navigateur pablo à l'adresse https://isn-flask.herokuapp.com/
Oui vous avez reconnu notre tp.

Pour héberger son site gratuitement on dispose de: * heroku qui necessite l'utilisation de *git* pour poser le code. * pythonanywhere
Je préfère ***heroku*** mais je conseillerai ***pythonanywhere*** pour les élèves d'isn pour sa simplicité d'utilisation.

1.5 5) Prolongement

- Utilisation d'une base de donnée
- Créer une image aux couleurs aléatoire (module pil) et l'afficher.

1.6 6) Exercices

1. - Créer un dossier ***static/file/*** et mettre dans ce dossier quelques fichiers textes (.txt) et images (.png ou .jpg).

   - Créer dans le fichier ***exemple.py*** une fonction ***listdir()*** qui retourne la liste les noms des fichiers contenus dans le dossier ***static/file/*** *(On utilisera la librairie* ***os*** *pour lister).*
   - modifier le ***templates/accueil.html*** pour faire afficher le nom des fichiers.

2. Modifier alors la fonction ***listdir()*** pour n'afficher que le nom des images.
3. Afficher cette fois les images dans la page internet.

```
In []:
```