1 Présentation de flask

1.1 Préliminaires

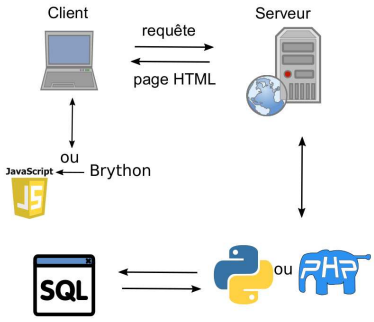
Classiquement sur internet on utilise un serveur ***lamp*** :
linux (os), **a**pache (serveur), **m**ysql (base de données), **p**hp (language de programmation pour avoir des pages dynamiques).
Il existe de nombreux outils pour le web écrit en python : *serveur Web* (Zope, gunicorn) ; *script cgi* ; *frameworks Web* (Flask, Django, cherrypy etc ...),
nous utiliserons le framework ***flask***.
Nous avons besoin de quelques fichiers pour commencer.
Télécharger le fichier isn-flask.zip et décompressez le dans le dossier *U :*
login
Ouvrez pyzo puis installez le module ***flask***

```
pip install  flask
cd U:\\login
```

1.2 Flask et les templates

a) Le principe

Lorsque vous saisissez une URL dans votre navigateur, que vous validez cette dernière, votre navigateur envoie une "requête" au serveur concerné afin qu'il nous renvoie une page web.
Tout d'abord, on nomme vulgairement l'échange de données entre votre navigateur et le serveur qui fournit les pages web un échange ***client / serveur***.
Le client représente votre navigateur.
Nous programmerons donc "coté serveur" en python.



Avec *Flask* on pourrait mettre le code dans un seul *fichier.py* mais on utilise de préférence des ***templates*** (avec ***jinja***).
L'arborescence d'un projet Flask sera :

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

Le dossier ***static/*** contiendra lui toutes les images, fichiers css, js ...
Les ***templates*** se mettent dans le dossier ***templates/*** et Le logiciel qui gère ces templates est ***jinja*** (on dit aussi moteur de templates).

b) Un exemple

Ouvrez avec *pyzo* le fichier ***isnflask.py*** :

```
1 #! python3
2 # -*- coding: utf-8 -*-
3 from flask import Flask, render_template, url_for
4 app = Flask(__name__)  # Initialise l'application Flask
5
6 @app.route('/')  # C'est un décorateur, on donne la route ici  "/"  l'adresse sera donc localhost:5000/
7 def accueil():
8     Lignes=['ligne {}'.format(i) for i in range(1,10)] # Que fait cette ligne ?
9     return render_template("accueil.html", titre="Bienvenue !",lignes=Lignes) # On utilise le template accueil.html, avec
      les variables titre et lignes
```

```
10
11 if __name__ == '__main__' :
12     app.run(debug=True)
```

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

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

.....

Exécuter le fichier ***exemple.py*** depuis *pyzo*, le server est lancé.
Ouvrez un navigateur internet à l'adresse `http://localhost:5000`.

Les templates sont des fichiers HTML dans lesquels on place des sections de code Jinja2, qui ressemble fortement à du Python.
Regardons maintenant fichier ***templates/accueil.html***.

```
1  <html>
2  <head>
3  <meta charset="utf-8" />
4  <title>ISN Pablo Neruda</title>
5  <!-- On importe notre frichier css -->
6  <link href="{{ url_for('static ', filename='mon_style.css') }}" rel ="stylesheet" type="text/css" />
7  <!-- On importe la librairie Brython -->
8  <script type="text/javascript" src="{{url_for('static', filename='brython.js') }}"></script>
9  <script type="text/javascript" src="{{url_for('static', filename='brython_stdlib.js') }}"></script>
10 </head>
11
12 <body onload="brython(1)">
13 <header class="site-header"><h1>{{ titre }}</h1></header> <!-- On affiche le titre -->
14 <section class="site-content">
15   <ul>
16     {% for ligne in lignes %} <!-- Avec jinja on peut utiliser des boucles des conditions ... -->
17       <li>{{ ligne }}</li>       <!-- On affiche le contenu de la liste ligne -->
18     {% endfor %}
19   </ul>
20 </section>
21 <footer>
22 {% include 'foot.html' %}
23 </footer>
24 </body>
25 </html>
```

Quelques explications sur le template :

- Tout ce qui est contenu entre ***{{ ..... }}*** ou ***{% ..... %}*** est exécuté par ***jinja***
  '{{ ..... }}' ne fait qu'afficher le contenu d'une variable, tandis que '{% ..... %}' sert à tout le reste.
- url_for('static', filename='mon_style.css') correspond donc à l'adresse du fichier /static/monstyle.css

1.3 Les images

Ajoutons une image. J'ai déjà mis deux images dans le dossier ***static/*** (favicon.ico et fleur.png).
Il suffit donc de rajouter dans ***templates/accueil.html***
Je rappelle la syntaxe pour les liens : ***href=" url_for('nom_du_dossier', filename='nom_du_fichier') "***

```
9  <script type="text/javascript" src="{{url_for('static', filename='brython_stdlib.js') }}" </script>
10 <!-- Les images -->
11 <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
12 </head>
13
14 <body onload="brython(1)">
15 <header class="site-header"><h1>{{ titre }} <img src="{{ url_for('static',filename ='fleur.png') }}" alt="fleur" title=
       "fleur" border="0"></h1></header>
```

Le ***favicon*** (ligne 12) est la petite icone que l'on voit dans l'onglet.

2 Les formulaires

2.1 Intérêt d'un formulaire

Le lecteur saisit des informations en remplissant des champs ou en cliquant sur des boutons, puis appuie sur un bouton de soumission (submit) pour l'envoyer soit à un script de page web dynamique tel que PHP, python etc... ou un script CGI.

2.2 Le principe

Les formulaires sont délimités par la balise ***<FORM> ... </FORM>***. Il est possible d'insérer dans une balise ***<FORM>*** n'importe quel élément HTML de base (textes, boutons, tableaux, liens,...) mais il est surtout intéressant d'insérer des éléments interactifs. Ces éléments interactifs sont :

- La balise ***<INPUT> :*** un ensemble de boutons et de champs de saisie
- La balise ***<TEXTAREA> :*** une zone de saisie
- La balise ***<SELECT> :*** une liste à choix multiples

Regarder ce que l'on peut faire avec la balise ***<input>*** : http://www.startyourdev.com/html/tag-html-balise-input
Nous avons avec ces exemples, obtenu des informations (texte, code couleur etc...) . La balise ***<form>*** sert à transmettre ces informations. Pour cela on doit indiquer une ***methode*** d'envoie des données et une ***action***

- ***METHOD*** indique sous quelle forme seront envoyées les réponses. Il y a deux méthodes ***GET*** ou ***POST*** ?
  Le choix de la méthode dépend de la façon dont les données sont reçues, de la taille et la nature des données.

- ***La méthode GET*** correspond à un envoi des données codées dans l'URL, et séparées de l'adresse du script par un point d'interrogation.
  Noter que lorsqu'on utilise le bouton retour, les requêtes GET sont exécutées à nouveau.
  Les données de formulaire doivent être uniquement des codes ASCII et la taille d'une URL est limitée à par le serveur.
  Par exemple https ://www.google.fr/ ?q=python utilise la méthode ***get*** avec la variable ***q*** (pour query). Testez ce lien dans un onglet.
  Si on modifie l'adresse (par exemple ***python*** en ***python+isn***), on modifie le contenu de la variabe ***q*** et donc la page html.
- ***La méthode POST*** correspond à un envoi de données stockées dans le corps de la requête.
- ***ACTION*** indique l'url de la page qui recevra et traitera les informations soumises. Cela peut aussi être l'envoie d'un mail

2.3 Exemple de formulaire utilisant la méthode POST

➔ Modifions la page ***templates/accueil.html*** pour mettre un *formulaire*
Ajouter les lignes suivantes

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

➔ Modifier maintenant le fichier ***exemple.py***.
On importe ***flask.request***

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

et on crée une nouvelle route.

```
9  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)
16 if __name__ == '__main__':
17     app.run(debug=True)
```

Le nom et le prenom sont envoyé au 2° template dans avec variables nom et prenom avec la méthode ***POST***.
On les affiche dans le template ***page2.html*** avec nom et prenom .
Regarder le code du deuxième template ***templates/page2.html***.

2.4 Ajouter Un lien qui ramène sur la première page

Vous connaissez tous les liens sur les pages *html*. Quand on clic dessus le lien vous renvoie sur une autre page.
La syntaxe est :

```
<a href="urldulien" > texte à afficher</a>
```

Par exemple pour créer un lien qui redirige vers mes cours on utilisera quelque chose comme ca.

```
<a  href="https://github.com/debimax/cours-debimax">Aller sur mon github</a>
```

Par contre la syntaxe pour rediriger vers une autre page de mon site est un peu différent. On utilise la fonction ***urlfor***.

➔ Éditez le fichier ***templates/page2.html***
Àjouter vers la fin du fichier

```
<a href="{{ url_for('accueil4') }}" >Formulaire avec redirection vers  la  même page</a>
```

accueil4 est le nom de la fonction python (*isnflask.py*) qui créer la page html.

2.5 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*

```
3  from flask import Flask, render_template, url_for, request
4  app = Flask(__name__) # Initialise l'application Flask
5
6  @app.route('/', methods=['GET','POST'])  # On doit indiquer que l'on utilise les deux méthodes
7  def accueil():
8      lignes=['ligne {}'.format(i) for i in range(1,10)]
9      try:
10         nom=request.form['nom']
11     except:
12         nom=''
13     try:
14         prenom=request.form['prenom']
15     except:
16         prenom=''
17     titre="Méthode {}".format(request.method)
18     return render_template("accueil.html", titre=titre,lignes=lignes,nom=nom,prenom=prenom)
19
20 if __name__ == '__main__':
21     app.run(debug=True)
```

et le template acceuil.html

```
22 </ul>
23 {% if  request.method == 'GET' %}
24     <div id="content">
25         <form method="post" action="{{ url_for('accueil') }}">
26         <label for="nom">Entrez votre nom:</label>
27         <input type="text" name="nom" /><br />
28         <label for="prenom">Entrez votre prénom:</label>
```

```
29          <input type="text" name="prenom" /><br />
30          <input type="submit" />
31          </form>
32          </form>
33      </div>
34  {% else %}
35      {% if  nom != '' or prenom != '' %}
36          <p>bonjour {{ prenom }} {{ nom }}</p>
37      {% endif %}
38  {% endif %}
39  </section>
```

3 Mettre son 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 nécessite 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.
Il y en a d'autres http://sametmax.com/quel-hebergement-web-pour-les-projets-python/

4 Documention

Pour se documenter je vous conseille ces quelques sites suivants.

- 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
- https://www.tutorialspoint.com/flask/flask_quick_guide.htm

5 Exercices

5.1 Exercice 1

En partant de l'exemple qui redirige vers la même page créer un site pour calculer l'indice de la masse corporelle (IMC).
On utilisera

- $IMC = \frac{masse}{taille^2}$ en $kg.m^{-2}$.
- Deux labels (masse et taille),
- Deux balises ***<input>*** de type number,
- Un bouton et un label pour afficher l'imc.
- On placera le tout avec un tableau pour faire simple.

5.2 Exercice 2

On ne souhaite pas afficher les images lorsque le client est un téléphone mobile.
Pour Flask ***request.user_agent*** détermine l'***useragent***.
Dans exemple.py, il faut modifier l'import du module Flask

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

Ajouter la fonction ***ismobile()*** que nous avons déjà utilisé pour brython.

```
9  def ismobile():
10     Agent=window.navigator.userAgent
11     if re.search('android|iphone|blackberry|symbian|symbianos|symbos|netfront|modelorange|
       javaplatform|iemobile|windows phone|samsung|htc|opera mobile|opera mobi|opera mini|presto|
       huawei|blazer|bolt|doris|fennec|gobrowser|iris|maemo browser|mib|cldc|minimo|semc-browser|
       skyfire|teashark|teleca|uzard|uzardweb|meego|nokia|bb10|playbook', Agent, re.IGNORECASE):
12         return True
13     else:
14         return False
```

Transmettre si c'est un mobile template html

```
15 return render_template("accueil.html", titre="Bienvenue !",lignes=Lignes,ismobile=ismobile())
```

Modifier alors le template ***acceuil.html*** pour ne pas afficher les images si c'est un mobile.
Pour vérifier si cela fonctionne avec le mobile on poura le remplacer ***ismobile=ismobile()*** par ***ismobile=True***

5.3 Exercice 3

1. - Créer un dossier ***static/file/*** et mettre dans ce dossier quelques fichiers textes (.txt) et images (.png ou .jpg).
   - À partir de la console pyzo esssayez de lister les noms des fichiers contenus dans un dossier quelconque *(On utilisera la librairie **os** pour lister).*
     Aide module OS : ce site ou le site w3big
   - Modifier le fichier ***exemple.py*** en ajoutant une fonction ***listdir()*** qui retourne la liste les noms des fichiers contenus dans le dossier ***static/file/*** .
   - 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 avec la balise <img />.