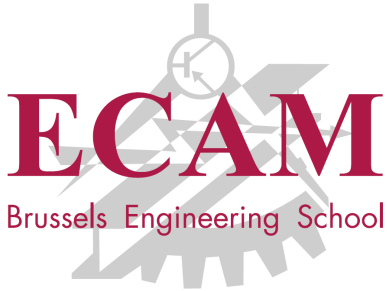


Développement Web avec Python

Sébastien Combéfis, Quentin Lurkin

2017–2018

Rappels

- Découverte du fonctionnement d'Internet et ses composants
  - Sortie du réseau local vers Internet avec le modem-routeur
  - Réseau de réseaux, adresses IP/MAC et routage
  - Fournisseur d'accès à Internet
- Communication et échanges sur le réseau Internet
  - Protocoles IP, DHCP, TCP, HTTP(S) et SMTP
  - Architecture client/serveur et serveur web
  - Internet des objets et cloud computing

Objectifs

- Développement d'un serveur web avec CherryPy
  - Prise en main du framework et premier site web
  - Définition de routes avec et sans paramètre
  - Templating avec jinja2
- Définition et traitement d'un formulaire

  Route avec le formulaire et route de traitement
- Définition d'une API REST d'interaction avec la BDD

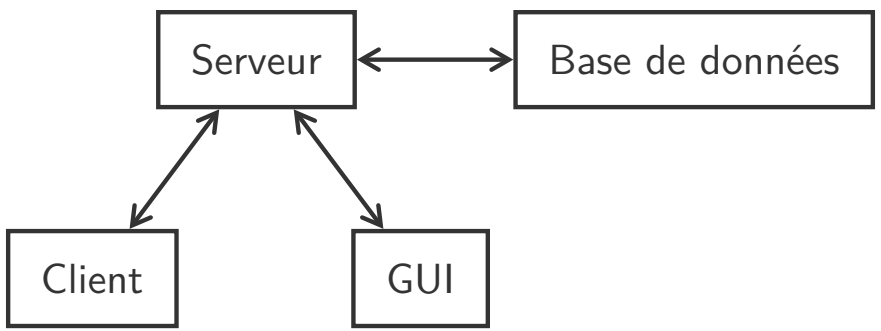
  Appel d'une API avec le module `urllib`

Vue globale

- Application web de gestion de quelque chose

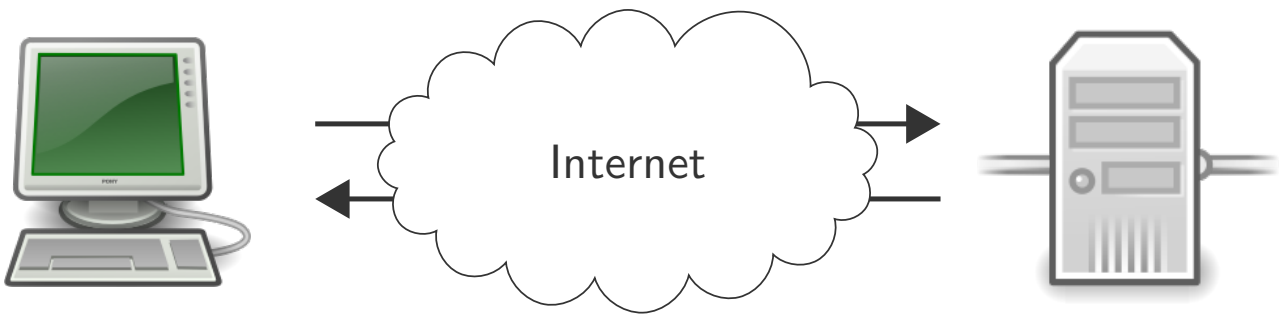
  Liste de courses, gestion des stocks de PQ, guide de voyage

- Application construite sur base de trois composantes
  - Serveur web en CherryPy
  - Interface graphique Kivy
  - Base de données en JSON



Client/Serveur

- Communication entre un client et un serveur
  1. Le client se connecte au serveur
  2. Le serveur accepte la connexion
  3. Le client et le serveur communiquent
     1. Le client envoie une requête au serveur
     2. Le serveur analyse la requête et répond au client
- La connexion peut être fermée par le client ou le serveur



CherryPy

Framework CherryPy

- Définition d'une nouvelle application `WebApp`

  Lancement de l'application avec la méthode `quickstart`

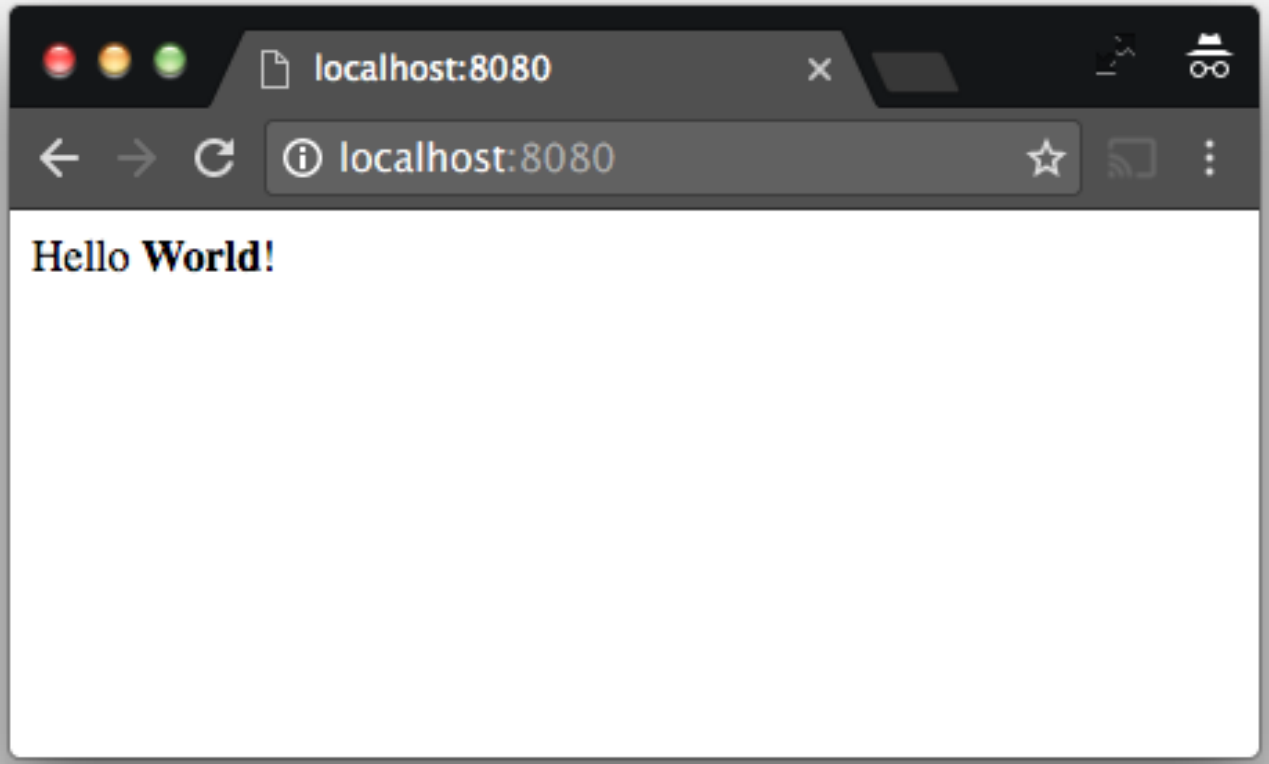
- Définition d'une nouvelle route `index`
  - Décorateur `@cherrypy.expose` pour chaque route désirée
  - Construction et renvoi d'un contenu HTML

```
1  import cherrypy
2
3  class WebApp():
4      @cherrypy.expose
5      def index(self):
6          return "Hello <b>World</b>!"
7
8  cherrypy.quickstart(WebApp())
```

Mon premier site web

- Site web lancé en local sur http://localhost:8080

  Accès à la route / qui appelle la méthode `index`



Route avec paramètre (1)

- Paramètres d'une route déclarés comme paramètre de fonction

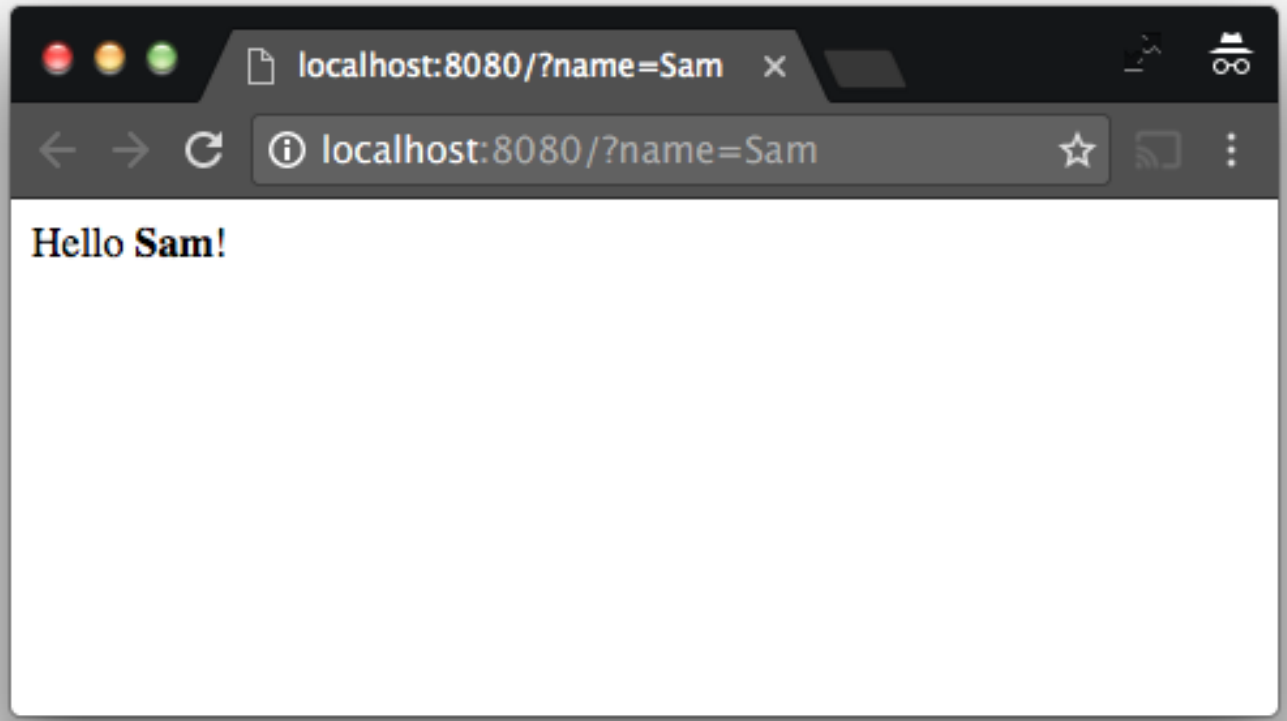
  Possibilité de définir une valeur par défaut

```
1  import cherrypy
2
3  class WebApp():
4      @cherrypy.expose
5      def index(self, name='World'):
6          return 'Hello <b>{}</b>!'.format(name)
7
8  cherrypy.quickstart(WebApp())
```

Route avec paramètre (2)

- Ajout des valeurs des paramètres après un ?

  Paramètres déclarés sous la forme de paires clé-valeur



Configuration

- Configuration du serveur dans un fichier texte séparé

  Déclaration du nom du fichier en créant l'application

- Configuration globale ou pour des ensembles de routes

```
[global]
server.socket_port: 9090
```

```
import cherrypy

class WebApp():
    @cherrypy.expose
    def index(self, name='World'):
        return 'Hello <b>{}</b>!'.format(name)

cherrypy.quickstart(WebApp(), '', 'server.conf')
```

Contenu statique (1)

- Séparation de la logique serveur du contenu HTML

  Fichiers `.htm` séparés importés avec méthode `serve_file`

```
1  Hello <b>World</b>!
```

```
1  import os
2
3  import cherrypy
4  from cherrypy.lib.static import serve_file
5
6  ROOT = os.path.abspath(os.getcwd())
7
8  class WebApp():
9      @cherrypy.expose
10     def index(self, name='World'):
11         return serve_file(os.path.join(ROOT, 'index.htm'))
12
13 cherrypy.quickstart(WebApp(), '', 'server.conf')
```

Contenu statique (2)

- Distribution automatique de contenu statique

  Configuration de routes distribuant le contenu d'un dossier

- Accès au contenu du dossier `public` par la route `static`

  Typiquement utilisé pour des images, fichiers CSS et JavaScript

```
1  [global]
2  server.socket_port: 9090
3
4  [/static]
5  tools.staticdir.on: True
6  tools.staticdir.root: os.path.abspath(os.getcwd())
7  tools.staticdir.dir: './public'
```

Template (1)

- Insertion de variables Python dans le contenu HTML

  La route doit renvoyer un dictionnaire Python

```
1  from datetime import datetime
2
3  import cherrypy
4  import jinja2
5
6  import jinja2plugin
7  import jinja2tool
8
9  class WebApp():
10     @cherrypy.expose
11     def index(self, name='World'):
12         return {'name': name, 'now': datetime.now()}
13
14 # [...]
```

Template (2)

- Enregistrement du moteur Jinja2 et configuration

  Se base sur les deux modules `jinja2plugin` et `jinja2tool`

```
1  # [...]
2
3  # Register Jinja2 plugin and tool
4  ENV = jinja2.Environment(loader=jinja2.FileSystemLoader('.'))
5  jinja2plugin.Jinja2TemplatePlugin(cherrypy.engine, env=ENV).
   subscribe()
6  cherrypy.tools.template = jinja2tool.Jinja2Tool()
7
8  cherrypy.quickstart(WebApp(), '', 'server.conf')
```

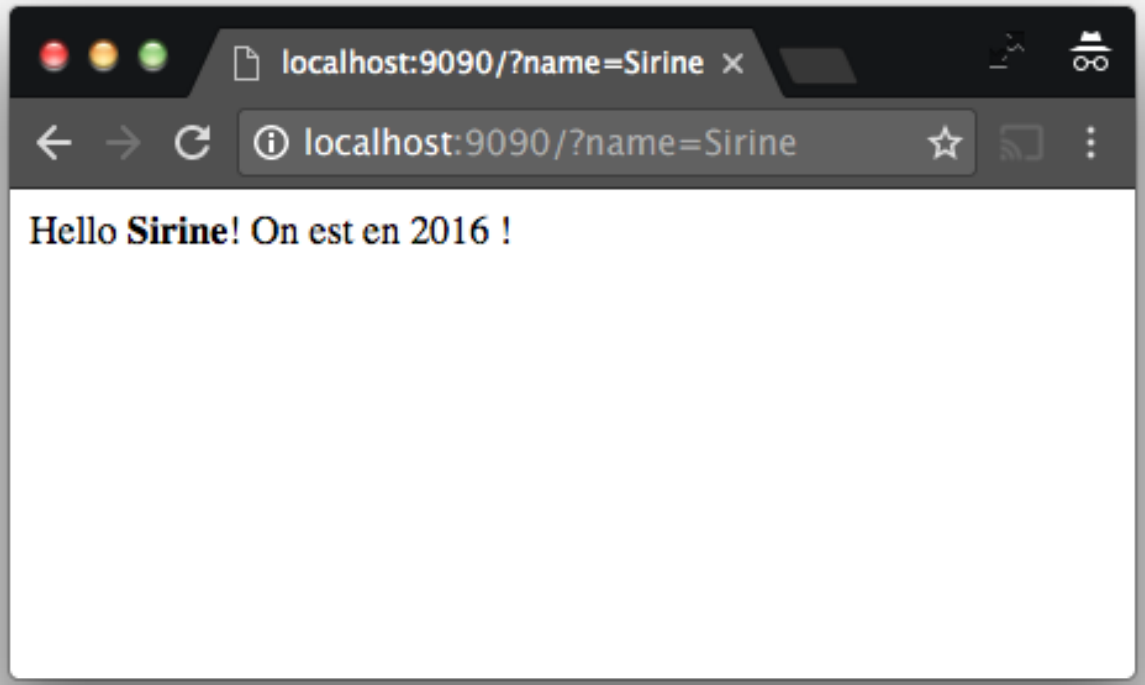
```
1  [global]
2  server.socket_port: 9090
3
4  [/]
5  tools.template.on: True
6  tools.template.template: 'index.htm'
7  tools.encode.on: False
8
9  # [...]
```

Template (3)

- Utilisation des variables dans le fichier HTML

  Insertion à l'aide d'une balise {{...}}

```
1  Hello <b>{{name}}</b>! On est en {{now.year}} !
```



Plusieurs routes

- Plusieurs points d'accès pour la même route

  Liste des URLs passée à la décoration `@cherrypy.expose`

```
1  from datetime import datetime
2
3  import cherrypy
4  import jinja2
5
6  import jinja2plugin
7  import jinja2tool
8
9  class WebApp():
10     @cherrypy.expose(['home', 'accueil'])
11     def index(self, name='World'):
12         return {'name': name, 'now': datetime.now()}
13
14 # [...]
```

Page d'erreur (1)

- Définition de la route par défaut

  Doit avoir un paramètre `attr` de valeur `abc`

- Renvoyer une chaine de caractères à convertir en binaire

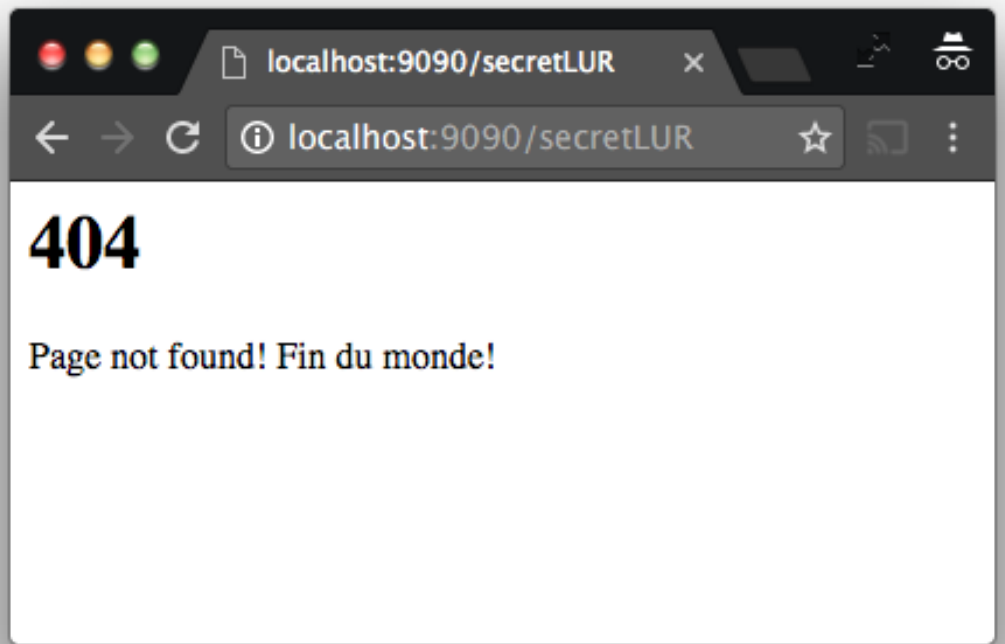
  Seulement lorsque `jinja2` est utilisé

```
1  # [...]
2
3      @cherrypy.expose
4      def default(self, attr='abc'):
5          return '<h1>404</h1><p>Page not found! Fin du monde!</p>'.
           encode('utf-8')
6
7  # [...]
```

Page d'erreur (2)

- Pages non existantes redirigées sur la route par défaut

  URL dont l'accès a été demandé stockée dans `cherrypy.url()`



Définir un formulaire (1)

- Une simple route qui renvoie le formulaire en HTML

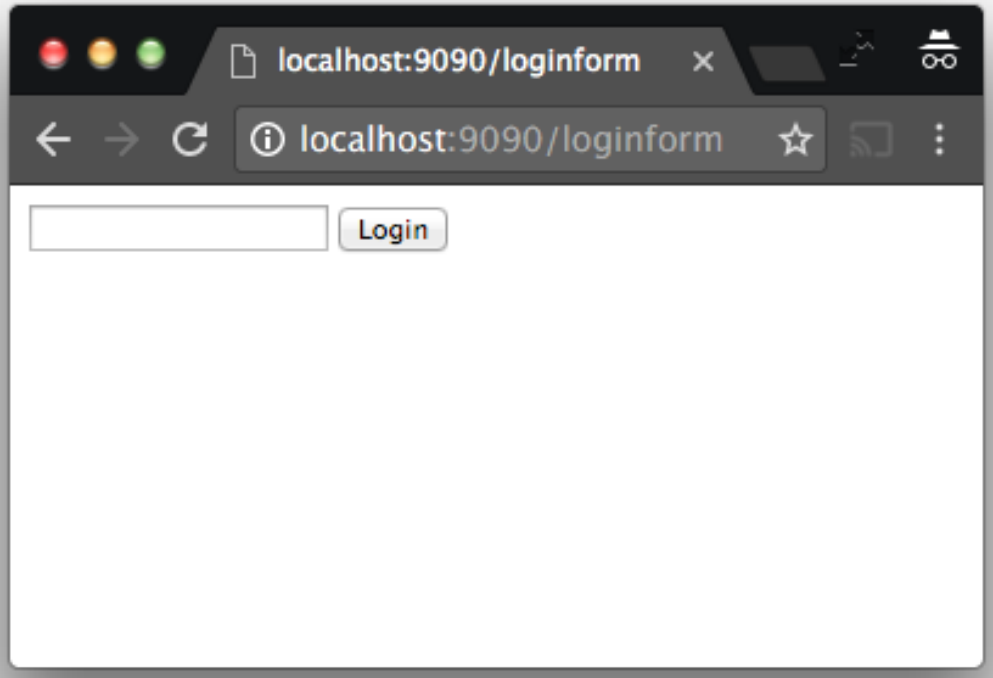
  Définition de la route de traitement via le paramètre `action`

```
1  # [...]
2
3      @cherrypy.expose
4      def loginform(self):
5          return '''<form action="/login" method="post">
6    <input type="text" name="name" />
7    <input type="submit" value="Login" />
8  </form>'''.encode('utf-8')
9
10 # [...]
```

Définir un formulaire (2)

- Bouton de type `submit` pour valider le formulaire

  Validation également lors de la pression de la touche `ENTER`



Traiter un formulaire (1)

- Définit une route de traitement du formulaire

  Nom de la route est celui déclaré dans la définition du formulaire

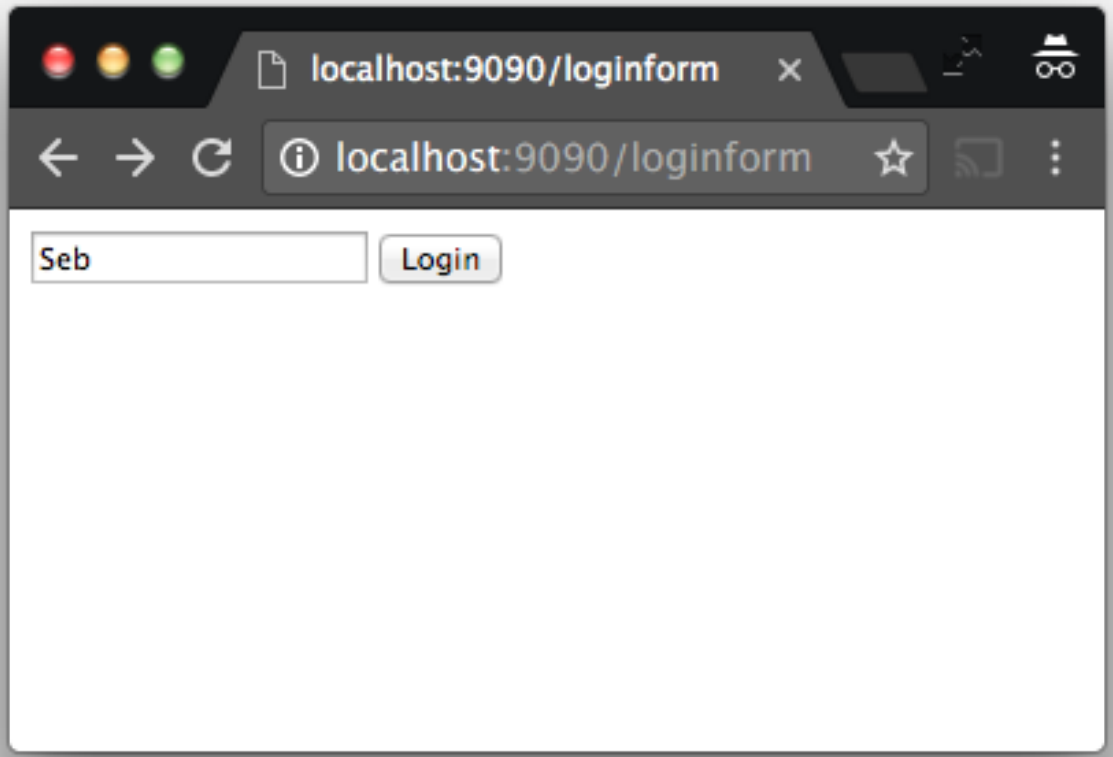
- Champs du formulaire reçu en paramètres de la route

  Mêmes noms que les propriétés `name` du formulaire

```
1  # [...]
2
3      @cherrypy.expose
4      def login(self, name):
5          return 'Bonjour {} !'.format(name).encode('utf-8')
6
7  # [...]
```

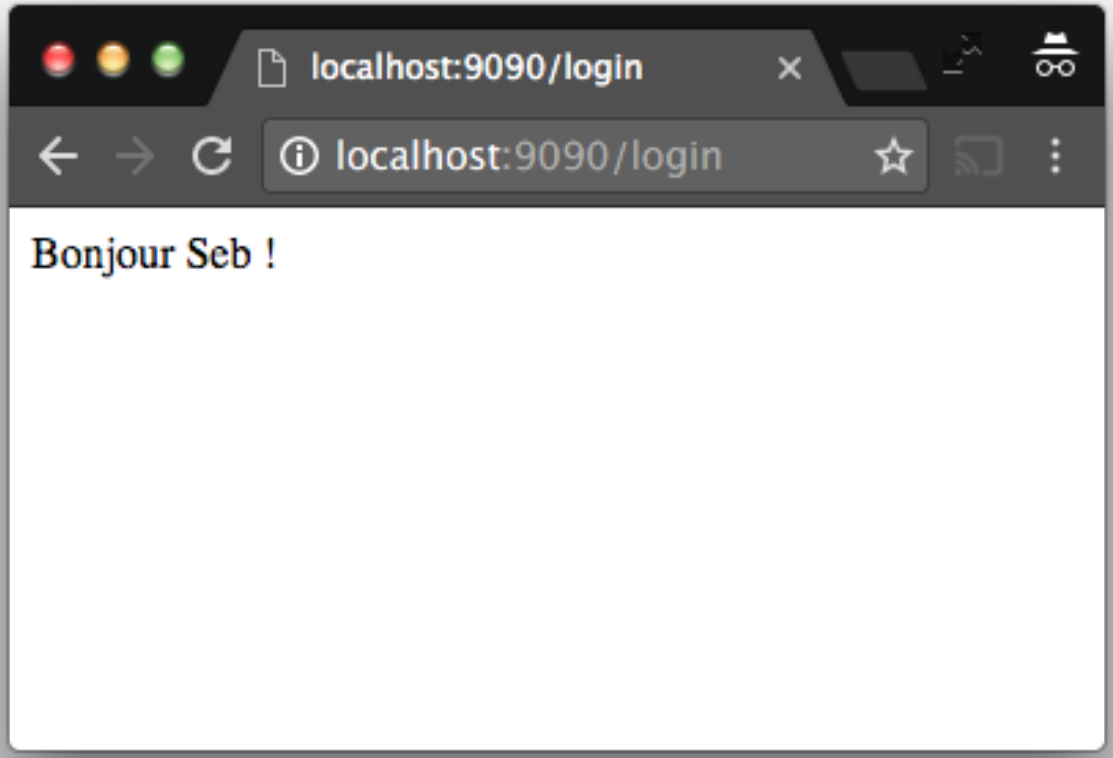
Traiter un formulaire (2)

- Encodage des données et validation du formulaire



Traiter un formulaire (3)

- Redirection sur la route de traitement du formulaire



Kivy et urllib

Route de type API

- Définition d'une route qui donne accès à des informations

  Format renvoyé par la route doit être choisi et défini

```
1  import json
2
3  # [...]
4
5      @cherrypy.expose
6      def listusers(self):
7          data = json.dumps({'users': ['CBF', 'LUR', 'VRL']})
8          return data.encode('utf-8')
9
10 # [...]
```

Interroger un serveur

- Chargement d'un URL avec la méthode `urlopen`

  À partir d'un objet `urllib.request`

```
1  import urllib.request
2  import json
3
4  url = urllib.request.urlopen("http://localhost:9090/listusers")
5  rawdata = url.read()
6  data = rawdata.decode('utf-8')
7
8  users = json.loads(data)
9  print(users)
```

```
{'users': ['CBF', 'LUR', 'VRL']}
```

Envoyer des données à un serveur (1)

- Construction d'un objet JSON avec `json.dumps`

  Encodage des données en URL avec `urllib.parse.urlencode`

- Envoi vers un URL sur le serveur avec `urlopen`

  À partir d'un objet `urllib.request`

```
1  import urllib.parse
2  import urllib.request
3  import json
4
5  user = {'name': 'Moi'}
6  data = json.dumps(user)
7  rawdata = urllib.parse.urlencode({'data': data})
8  urllib.request.urlopen("http://localhost:9090/adduser?" + rawdata)
```

Envoyer des données à un serveur (2)

- Définition d'une route de traitement côté serveur

  Récupération du JSON avec `json.loads`

- Renvoi d'une valeur qui peut être vérifiée par l'appel

```
1  # [...]
2
3      @cherrypy.expose
4      def adduser(self, data):
5          user = json.loads(data)
6          print('> Ajout de ' + user['name'])
7          return 'OK'.encode('utf-8')
8
9  # [...]
```

Crédits

- https://openclipart.org/detail/180746/tango-computer-green
- https://openclipart.org/detail/36565/tango-network-server
- https://www.flickr.com/photos/fearless_craig/3997311986
- https://www.flickr.com/photos/oweniverson/2327608121