

4TPM207U / Système de Gestion de Base de Données (Coloration informatique 2016/2017)

4TPM207U / Système de Gestion de Base de Données (Coloration informatique 2016/2017)

› TD Machines › TP Flask

TP Flask

- Soyez sûr d'avoir configuré l'environnement pour faire fonctionner le code Python Flask, comme expliqué sur la page dédiée à ce sujet sur le cours.
- L'utilisation du module wsgi empêche Flask d'afficher les erreurs de compilation Python. Si une telle erreur se produit, vous n'aurez à l'écran qu'un message du type "**Internal server Error**", sans précision.

Pour pouvoir voir les erreurs de compilation de votre fichier python webapp.py, et le résultat des commandes print de python, il faut taper les commandes suivantes dans une fenêtre Terminal:

1. `cd ~/espaces/www/flask-scripts`
2. `export FLASK_APP=webapp.py`
3. `export FLASK_DEBUG=1`
4. `flask run`

Quand le serveur vous dit qu'il est prêt (c'est-à-dire quand le message ' * Debugger is active!' s'affiche), connectez-vous à `http://localhost:5000`.

A chaque fois que vous rafraîchissez cette page `http://localhost:5000`, les erreurs de compilation éventuelles s'affichent dans la fenêtre Terminal (et dans votre navigateur selon le type d'erreur), ce qui vous permettra de les corriger plus facilement.

- Ouvrez le fichier `webapp.py` dans un éditeur de texte. Pouvez-vous deviner comment exécuter la fonction `hello_name()` en changeant l'url ?

Administration

Administration page



Paramètres



Rôles attribués localement



Permissions



Voir les permissions



Filtres



Historiques



Sauvegarde



Restauration

Administration du cours

Prendre le

- Nous allons transformer la route racine (méthode hello()) pour qu'elle affiche un formulaire HTML permettant la saisie d'un prénom et qui appellera, une fois validé, la route hello_name(name). Plusieurs façons de faire cela sont possibles, nous allons en voir une en particulier.

rôle...

Administra
tion du site

1) Créez d'abord un répertoire appelé 'static' dans votre répertoire flask-scripts. Ce répertoire va contenir un fichier html qui lui-même contiendra notre formulaire.

2) Créez donc dans ce répertoire 'static' un fichier appelé pour cet exemple 'form.html' . Ce fichier devra contenir le code html suivant:

```
<form method="POST" action="after_form">
  <input type="text" name="prenom" placeholder="*Prénom*" />
  <input type="submit" value="Soumettre" />
</form>
```

Pouvez-vous comprendre ce code ?

3) Nous voulons que ce formulaire soit affiché dans la page web quand le navigateur suit la route '/'. Pour cela, il faut modifier la fonction hello dans webapp.py en :

```
@app.route("/")
def hello():
    return app.send_static_file("form.html")
```

Comprenez-vous pourquoi ?

4) La partie ' action="after_form" ' du code dans form.html fait que la route after_form est suivie par le navigateur une fois que l'utilisateur a validé le formulaire. Pour pouvoir utiliser le prénom entré par l'utilisateur, on peut ajouter dans webapp.py le code suivant:

```
@app.route('/after_form', methods=['POST'])
def after_form():
    print("I got it!")
    return hello_name(request.form['prenom'])
```

Testez vos modifications. Comprenez-vous le mécanisme en détail ?

Plus de détails sur ce mécanisme peut être trouvé sur, par

exemple, cette page.

- On voudrait maintenant que le prénom saisi soit celui d'un dragon utilisé pendant le TP SQL, et que notre site affiche des informations sur ce dragon.

1) Assurez-vous tout d'abord que votre base de données du TP soit correcte. Au besoin, vous pouvez utiliser ce fichier pour créer la base.

2) Rajoutez la fonction suivante à votre fichier webapp.py:

```

import psycopg2
def display_dragon(prenom_dragon):
    # Try to connect to an existing database
    print('Trying to connect to the database')
    try:
        conn = psycopg2.connect("host=dbserver dbname=U
SERNAME user=USERNAME")
        print('Connected to the database')
        cur = conn.cursor()
        command = 'select * from dragons where dragon =
\'\' + prenom_dragon + \'\' ;'
        print('Trying to execute command: ' + command)
        try:
            # Query the database and obtain data as Pytho
n objects
            cur.execute(command)
            print("execute ok")
            #retrieve all tuple
            rows = cur.fetchall() #rows => tableau (les l
ignes du résultat) de listes (les différents attrib
uts du résultat)
            print("fetchall ok")

            page = ''
            dragon = rows[0]
            page = prenom_dragon
            if dragon[1] == 'M':
                page = page + ' est un male'
            else:
                page = page + 'est une femelle'
            page = page + ' et a ' + str(dragon[3]) + '
ecailles.\n'

            # Close communication with the database
            cur.close()
            conn.close()
            print('Returning page ' + page)
            return page
        except Exception as e :
            return "error when running command: " + comma
nd + " : " + str(e)
        except Exception as e :
            return "Cannot connect to database: " + str(e)

```

- 3) Attention, il faut changer (deux fois) USERNAME sur la cinquième ligne par votre login.

4) Comprenez-vous cette fonction ?

5) Comment faut-il changer la fonction after_form pour

qu'elle appelle cette nouvelle fonction `display_dragon` plutôt que la fonction `hello_name` ?

6) Testez vos modifications, en entrant par exemple Smeagol comme nom de dragon.

- Changez la fonction `display_dragon` pour qu'elle affiche le régime alimentaire du dragon et le nom de ses prétendants. (Attention quand vous tapez votre code si vous utilisez un éditeur de texte qui ne gère pas correctement les indentations Python: n'utilisez pas la touche Tab pour l'indentation mais des caractères espace)
- Nous allons maintenant ajouter une gestion d'erreur complète si le nom du dragon saisi n'existe pas dans la base de données. Pour l'instant, vous avez simplement un message d'erreur qui est affiché dans le navigateur par la ligne

```
return "error when running command: " + command + "  
: " + str(e)
```

1) Au lieu de renvoyer le message d'erreur, nous allons rediriger l'utilisateur vers le formulaire de saisie pour que celui-ci affiche le message d'erreur si nécessaire puis le formulaire de saisie.

Pour commencer, il faut mettre en place la redirection vers la route '/' qui est gérée par la méthode `hello()`. La ligne devient donc :

```
return redirect url_for('hello', error=str(e))
```

Quel est l'intérêt d'utiliser la commande `url_for()` qui prend en paramètre le nom d'une méthode Python au lieu de l'URL de la route ?

2) La méthode `hello` prend maintenant un argument optionnel. La déclaration de la méthode devient donc :

```
def hello(error=None):
```

Ceci est utile si vous tentez d'accéder à la route directement depuis votre navigateur sans passer par le formulaire. Si vous essayez votre code maintenant, au lieu du message d'erreur, vous avez le formulaire de saisie et plus de message d'erreur 😞

3) Pour avoir le message d'erreur, il faut transformer le formulaire en "template html" et utiliser la commande `render_template` de Flask. Les templates sont à déposer dans un répertoire 'templates' comme pour le dossier 'static' créé plus tôt.

4) Copiez votre fichier form.html dans le répertoire templates. Votre route '/' devient donc:

```
@app.route("/")
def hello(error=None):
    return render_template("form.html", hasError=error)
```

5) Il faut maintenant ajouter la gestion d'erreur dans le fichier form.html :

```
{%if hasError %} <p><strong><font color="red"> {{hasError}} </font></strong></p> {%endif%}
```

6) Essayez ! Dans notre embryon d'application web, nous venons de mettre en place **une bonne pratique qui consiste à séparer le fond** (récupération des données, traitement des données souvent appelé couche logique) **de la forme** (l'affichage des données au client).

- Une autre bonne pratique est d'éviter de dupliquer des informations. Par exemple, actuellement dans nos fichiers le nom de la route appelée au moment de valider le formulaire est écrit plusieurs fois : dans la définition de la route et dans la balise *action* du formulaire html. Si vous décidez de changer le nom de la route de la ligne

```
@app.route('/after_form', methods=['POST'])
```

de `after_form` en `validate`, vous constatez que la validation du formulaire ne fonctionne plus. L'URL qui est appelée après la validation du formulaire n'existe plus. Vérifiez.

- Il est nécessaire de changer cette information dans le formulaire html. Pour éviter cette source d'erreur classique (imaginez un site avec plusieurs dizaines de formulaires), il ne faut pas dupliquer l'information. La commande `url_for()` va nous aider car elle peut aussi être utilisée dans les templates html. Modifiez la balise *action* de votre formulaire html :

```
<form method="POST" action="{{url_for('after_form')}}">
```

- Changez le nom de la route sur la ligne

```
@app.route('/after_form', methods=['POST'])
```


et vous verrez que le code html généré pour le formulaire sera correct.

- Nous allons maintenant faire un jeu !
 - En utilisant la balise 'select' du langage HTML, ajoutez

dans votre formulaire html une liste pour choisir le nom d'un dragon. Une bonne façon de faire pour séparer le fond de la forme est de construire votre requête SQL et d'envoyer son résultat tel quel au template HTML. Vous utiliserez une boucle 'for' dans le template HTML pour construire les différents choix possible de la balise 'select'.

- A partir d'un choix A effectué dans la liste et du nom B d'un dragon saisie, faites afficher GAGNE (ou une image qui représente une médaille par exemple) si le dragon A aime le dragon B.

Modifié le: mercredi 12 avril 2017, 21:16

 Documentation Moodle pour cette page