TP : Pokemon & Langage serveur

I. Mise en place à partir d'un projet existant

Dans cette activité, nous allons réutiliser les formulaires vus dans le cours sur les serveurs pour créer un générateur de Pokémon minimaliste.
Nous allons travailler sur 3 points importants :

- gestion des formulaires et calculs côté serveur
- gestion de page statique sur les serveurs
- gestion de cookies (et de sessions)

❐ — Exercice 1 —

- Téléchargez l'archive zip "Pokémon" depuis bouillotvincent.github.io . Ce site est un mélange d'un exercice réalisé au Chapitre 4 et à la dernière séance. Il va être la base de notre site web généré côté serveur. Dans la foulée :
  - ➡ renommez views.py en application.py ;
  - ➡ credentials.html est mal placé et ne fonctionnera pas. Placez-le avec les autres templates. Il servira pour la gestion des cookies.

À ce stade, vous avez donc l'arborescence suivante :

```
.
└── flask
    ├── application.py
    └── templates
        ├── index.html
        ├── credentials.html
        ├── monPokemon.html
        └── resultat.html
```

❐ — Exercice 2 —

Nous allons faire le ménage dans les différents fichiers afin que l'on se retrouve avec un code minimal fonctionnel. Les instructions sont ici volontaires vagues et doivent vous amener à vous faire réfléchir sur les différents éléments nécessaires à la réalisation d'une opération donnée.

Attention : Pour éviter de consommer trop de ressources, votre navigateur met en cache les sites web que vous visitez… même localhost!
Pour éviter cela, je vous conseille d'utiliser **FIREFOX** ou **CHROME** en mode **NAVIGATION PRIVEE** pour tous nos tests à partir de maintenant.

- Dans **monPokemon.html :**
  - il va être impossible d'obtenir le surnom secret du pokémon. Ajoutez un attribut **name** au bon endroit afin de récupérer le "**surnom**" de votre Pokémon (souvenez-vous de la manière dont nous avons traité l'envoi des variables vers le serveur Flask avec **index.html)**
  - ligne 14 : il manque une instruction essentielle pour envoyer votre formulaire. La rajouter (voir **index.html** si besoin).

- Dans **application.py :**
  - nettoyez la fonction `index` de manière à ce qu'une requête HTTP à localhost:5000/ renvoie la page monPokemon.html .
  - si besoin, corrigez les lignes suivant result = request.form. On veut récupérer les nom et surnom à partir du fichier monPokemon.html et l'envoyer vers la page resultat.html.
  - la page obtenue à l'adresse localhost:5000/about est inutile. Supprimez-la.

- Dans **resultat.html :**
  - vous voulez récupérer le surnom, pas le prénom !
  - ajoutez un lien vers la racine du serveur '/' . Le texte du lien sera "Accueil".

Vous devez disposer à présent d'un exemple minimal fonctionnel permettant de faire fonctionner les deux premiers boutons de notre formulaire. Dans la suite, pour rester le plus clair possible, nous nous restreindrons à ces deux "inputs". Le principe serait le même si l'on souhaitait inclure les boutons radios et les checkbox.

II. Calculs côté serveur, création d'une page de résultat

L'intérêt d'avoir un serveur est de pouvoir lui demander de faire des calculs complexes : je vous demanderais de le faire pour le convertisseur IEEE754.

Ici, à partir des données du formulaires, nous allons faire de petits calculs afin de déterminer les points de vie (**pLife**) et les points de force (**pForce**) de notre Pokémon.

Formules :
pVie = nombre aléatoire entre 100 (exclus) et 150 si le nom du Pokémon a plus de 5 lettres strictement et, sinon, entre 50 et 100.
pForce est égale à une fonction du **nombre de lettre du surnom du Pokémon**. Cette fonction vaut $f(x) = \lfloor x^2 - 4x + 8 \rfloor$ avec $\lfloor$ et $\rfloor$ les opérateurs d'arrondi à la valeur inférieure.

❐ — Exercice **3** —

Dans **application.py** :

- créez deux fonctions calculerVie et calculerForce qui prennent en paramètre un entier x et renvoient les valeurs de pVie et pForce. On aura sans doute besoin d'importer une librairie Python…
- dans la fonction resultat(), créez des variables appelées pVie et pForce. Donnez-leur pour valeur le résultat des fonctions calculerVie(len(result['nom'])) et calculerForce(len(result['surnom'])), puis ajoutez deux paramètres pdv = pLife et pdf = pForce dans render_template afin de les envoyer à résultat.html.

Dans **resultat.html :**

- Modifiez votre fichier résultat.html de telle manière à obtenir l'affichage ci-contre.

Il est évident que l'on pourrait faire des calculs beaucoup plus complexes, gérer des évolutions de Pokémon, gérer un parc de Pokémon à partir d'une base de données ou créer des "aventures". Vous êtes libres de tester les potentialités de votre "serveur" local…

III. Gestion des pages et objets statiques

Jusqu'à maintenant, nous avons discuté de l'intérêt du dynamisme sur le web. Il est évident que tout n'est pas dynamique sur un site web ! En particulier, les CSS, les polices spécifiques au site et les images ne sont pas dynamiques. Elles sont dites **statiques**.

❐ — Exercice **4** —

Dans le framework Flask de Python, les fichiers statiques sont regroupés dans un dossier appelé "**static**" qui se trouve directement dans le dossier flask.
Créez ce dossier puis déplacez les fichiers de style dans ce répertoire "**static**".

Si vous essayez de voir le résultat, vous allez constater que rien n'a changé : les fichiers CSS et les polices Pokemon ne s'ouvrent pas…
Nous allons devoir jouer un peu avec Jinja 2 pour y parvenir.

❒ — Exercice **5** —

- Il va falloir modifier complètement la ligne de chargement du css : quelle ligne permet de faire cela dans les fichiers HTML ?

- Remplacez-là par :
  <link href="{{ url_for('static',filename='style.css') }}" rel="stylesheet" type="text/css"/>

- Redémarrez votre serveur et testez votre site Internet.

Explication : url_for est une commande Jinja2 permettant de retrouver un fichier grâce à son nom à un endroit de l'arborescence (ici static).

IV. Voulez-vous des Cookies ?

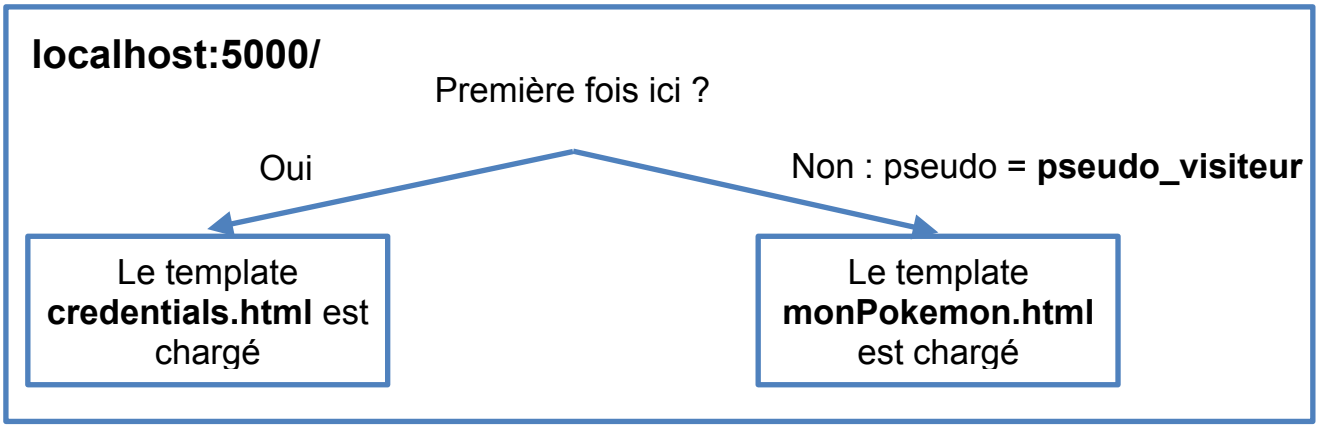
Notre page donne déjà plus envie de créer des Pokemon… Un bon CSS permet de vraiment mettre en valeur une page web.

Essayons à présent de mettre en place une "session" sur la base de cookies.

Un cookie, c'est quoi ? On entend trop souvent dire que c'est le mal… Mais non! C'est juste un petit fichier (moins de 4ko) qui contient des informations vous facilitant (théoriquement) la vie.

Dans notre petite application Pokemon, nous allons savoir si un utilisateur s'est déjà connecté en vérifiant si un cookie existe ou non.
À la première connexion, le site web vous demandera d'écrire un pseudo : un cookie contenant une variable appelé *pseudo* sera alors écrit sur votre disque dur.
Lors des connexions suivantes, l'utilisateur sera ensuite reconnu par son pseudo !

Cela va donner la structure schématique suivante :



Pour récupérer le contenu du cookie pseudo, l'instruction en Python Flask est :
pseudo_visiteur = request.cookies.get('pseudo')
Si le cookie n'existe pas, **None** est renvoyé.

❐ — Exercice **6** —

Dans application.py :

- Importez les "fonctions" **make_response** et **redirect** depuis la librairie flask.
- Modifiez la fonction index() afin qu'un test sur l'existence de la variable pseudo_visiteur soit fait. On cherchera si la variable est "**None**" ou non et on enverra l'utilisateur vers la page credentials.html ou monPokemon.html en fonction.
- On passera **"pseudo=pseudo_visiteur"** dans l'appel à render_template qui nous intéresse.

Dans monPokemon.html :

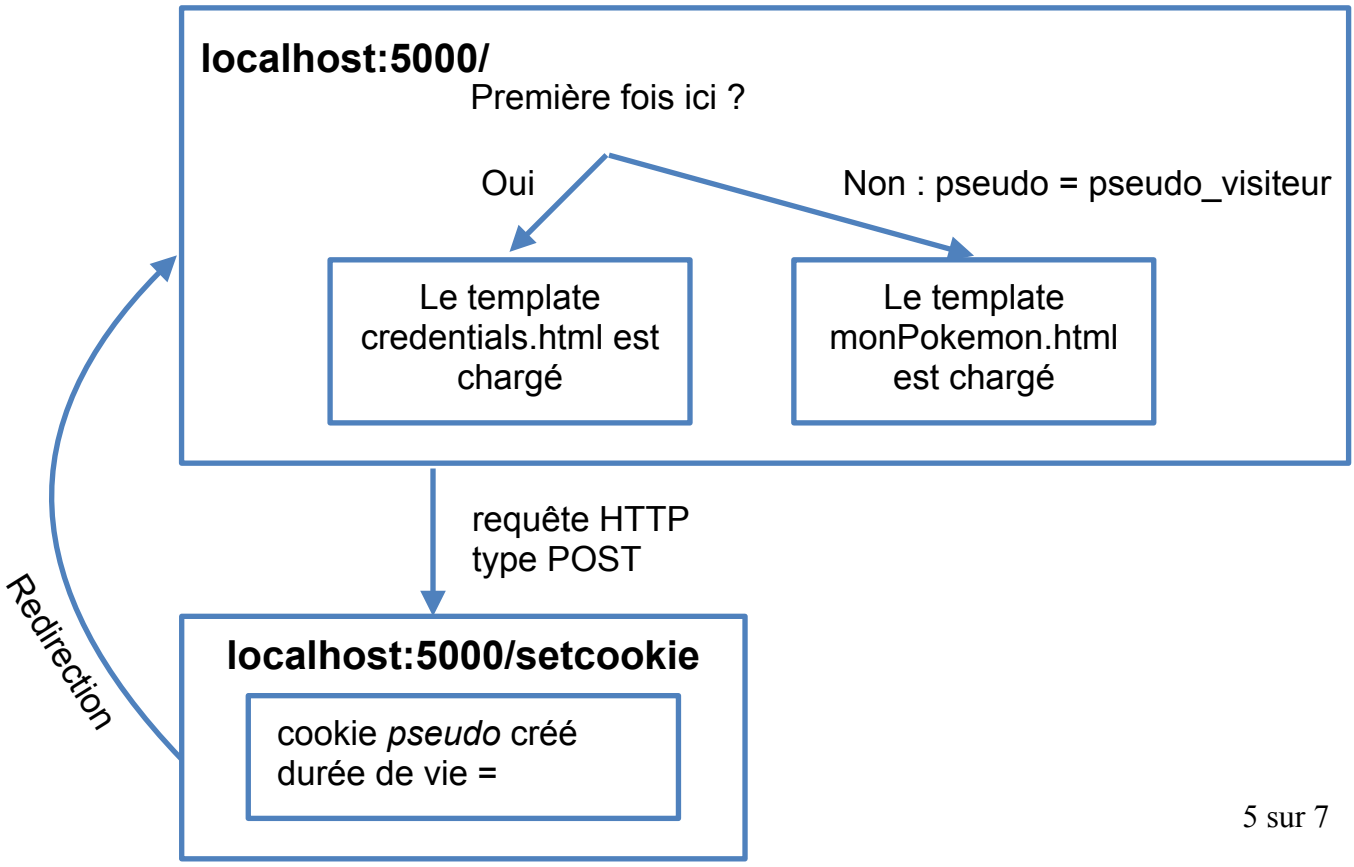
- Remplacez la ligne 12 par :
  <h1> Bienvenue {{ pseudo }}, créez le pokémon de vos rêves! </h1>

Pour l'instant, lorsque vous testez votre code, cela doit lamentablement planter.
Pourriez-vous expliquer rapidement pourquoi ?

_____

_____

Reprenons la structure de notre site web pour comprendre :



❐ — Exercice **7** —

Dans application.py :

❖ Copiez les instructions suivantes :

```
@app.route('/setcookie',methods = ['POST','GET'])
def rerouting():
    if request.method == 'POST':
        result = request.form
        reponse = make_response(redirect('/'))
        reponse.set_cookie('pseudo', result['pseudo'], max_age=3600*24)
        return reponse
    else:
        return redirect('/')
```

❖ Sur une page au format A4, reprenez le diagramme ci-dessus et indiquez quelles instructions sont utilisées dans la fonction rerouting() pour la redirection et la requête.

❖ Quelle est la durée de vie de notre cookie ? _____
Quelle est l'utilité du else à la fin de la fonction ?

_____

_____

_____

❖ Que se passe-t-il si vous tapez localhost:5000/setcookie ?

Nous avons ici crée un cookie avec une durée de vie limitée. Certains cookies peuvent être paramétrés pour avoir une durée de vie illimitée.

Mais, où est donc notre cookie ????

Dans chrome, vous pouvez le retrouver ici : chrome://settings/content/cookies?search=cookie
Dans les autres navigateurs, vous pouvez le voir/supprimer dans les options pour développeurs.

❐ — Exercice **8** —

❖ Supprimer votre cookie comme indiqué ci-dessus et rafraîchissez votre page web. Que constatez-vous ?

Conclusion sur les cookies:

Non les cookies ne sont pas le mal et sont plutôt utiles pour stocker des choses. Par contre, ils ont 4 problèmes principaux :

(I) ils ne sont pas sécurisés : les données sont visibles à TOUS !!

(II) les navigateurs peuvent désactiver les cookies. Si votre site web dépend fortement de ceux-ci, cela risque de poser problème chez certains utilisateurs.

(III) les navigateurs limitent le nombre de cookies à 30 environ et 4ko par cookie.

(IV) les cookies sont envoyés à chaque fois depuis votre ordinateur vers le serveur. 20 cookies x 4 ko = 80ko en upload à chaque connexion…

❐ — Exercice **9** —

- Complétez le diagramme représentant votre site web en introduisant les pages relatives au résultat ainsi que toutes les requêtes associées.