TP : Pokemon & Langage serveur

I. Introduction

Dans cette activité, nous allons réutiliser les formulaires vus dans le cours du côté de Flask 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)

— À faire vous-même **1** —

- Copier l'exercice 4 et nommer la copie "Pokemon". Ceci va être la base de notre site web généré côté serveur. Dans la foulée :
  - renommez views.py en application.py ;
  - téléchargez credentials.html, pour mettre les cookies au four.
- Récupérer le fichier corrigé monPokemon.html de l'exercice 2 (formulaire sur la création d'un Pokemon) et le coller dans flask/templates.

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

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

— À faire vous-même **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.

- Dans **monPokemon.html :**
  - il existe un attribut **name (**voir **index.html** si besoin**)** qui n'apparait pas dans les lignes 17 et 21
  - sur ces lignes, ajouter un attribut "nom" et "surnom" à la balise input.`
  - 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 telle manière à ce qu'une requête HTTP à localhost:5000/ renvoie la page monPokemon.html .
  - corrigez les lignes 14 et 15 afin de récupérer et d'envoyer la variable renommée dans le fichier **monPokemon.html** .
  - la page obtenue à l'adresse localhost:5000/about n'est plus obligatoire .

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

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

Nous disposons à présent d'un exemple minimal fonctionnel permettant de faire fonctionner les deux premiers boutons de notre formulaire. Dans la suite, pour des raisons de clarté, nous nous restreindrons à ces deux "inputs" mais le principe serait le même si l'on souhaitait inclure les boutons radios et les checkbox.

L'intérêt d'avoir un serveur est de pouvoir lui demander de faire des calculs complexes. À partir des données du formulaires, nous allons donc calculer les points de vie (pLife) et les points de force (pForce) du Pokémon que nous souhaitons créer.

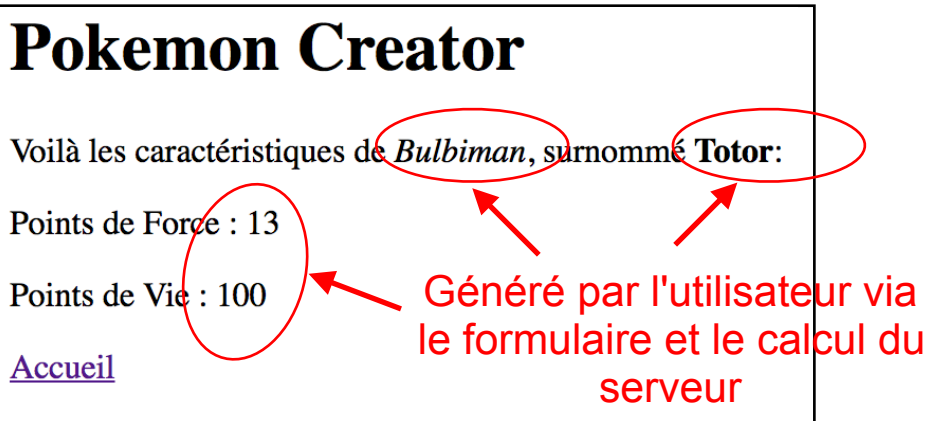
Formules :
pLife = nombre aléatoire entre 100 (exclus) et 150 si le nom du Pokémon a plus de 5 lettres strictement et, sinon, entre 50 et 100.
pForce = f(nombre de lettre du surnom du Pokémon) avec $f = \lfloor x^2 - 4x + 8 \rfloor$ avec $\lfloor$ et $\rfloor$ les opérateurs d'arrondi à la valeur inférieure.

❐ — À faire vous-même **3** —

Dans **application.py** :
- créez deux fonctions computeLife et computeForce qui prennent en argument un entier et renvoient les valeurs de pLife et pForce. On aura sans doute besoin de librairies Python supplémentaires…
- dans la fonction resultat(), attribuez aux variables pLife et pForce le résultat des fonctions computeLife(len(result['nom'])) et computeForce(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 en parc de Pokémon 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**.

❐ — À faire vous-même **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. Téléchargez les fichiers proposés dans la Séance 5, **Divers**, dézippez-les et déplacez-les dans le répertoire "**static**" nouvellement créé.

A ce stade, les fichiers CSS et les polices Pokemon ne doivent pas s'ouvrir…
Nous allons devoir jouer un peu avec Jinja 2 pour y parvenir.

❐ — À faire vous-même **5** —

Il va falloir modifier complètement la ligne de chargement du css (ligne 7 de monPokemon.html) dans tous les fichiers HTML

<link href="{{ url_for('static',filename='style.css') }}" rel="stylesheet" type="text/css"/>
Redémarrez votre serveur.

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

Remarque importante : Lorsque vous redémarrez votre serveur, le CSS ne devrait pas fonctionner. C'est normal ! 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 **CHROME en mode NAVIGATION PRIVEE** pour tous nos tests à partir de maintenant.

IV. Voulez-vous des Cookies ?

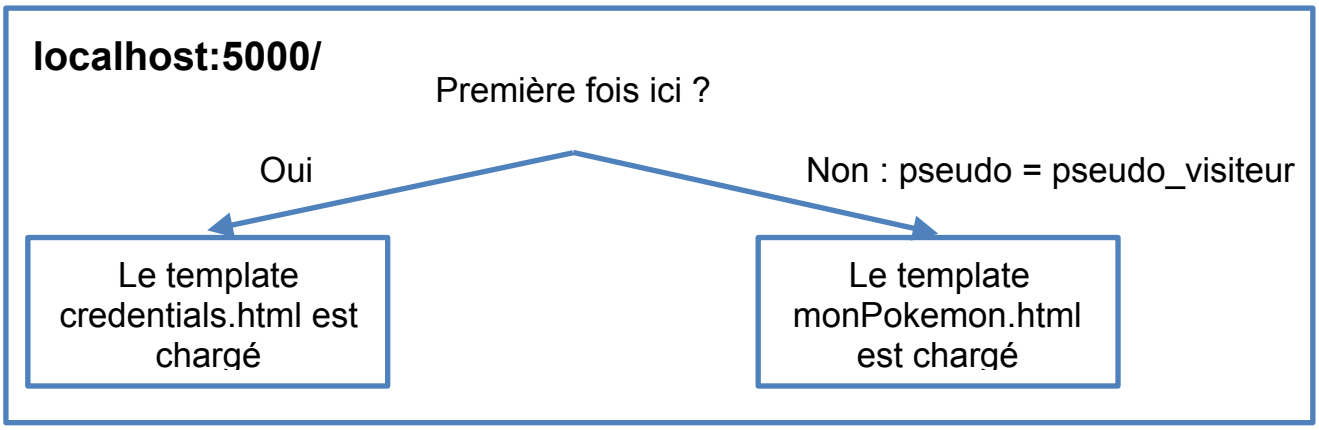
Notre page donne déjà plus envie de créer des Pokemon… Un bon CSS permet de vraiment mettre en valeur une page web.

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, un cookie contenant une variable appelé *pseudo* sera donc créé en demandant le prénom (*pseudo*) de l'utilisateur. L'utilisateur sera ensuite reconnu par son prénom lorsqu'il se connectera les fois suivantes.

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')

❐ — À faire vous-même **6** —

Dans application.py :

- Importez 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 fera attention à bien passer pseudo=pseudo_visiteur en paramètre lorsque l'on va sur la page monPokemon.html.

Dans monPokemon.html :

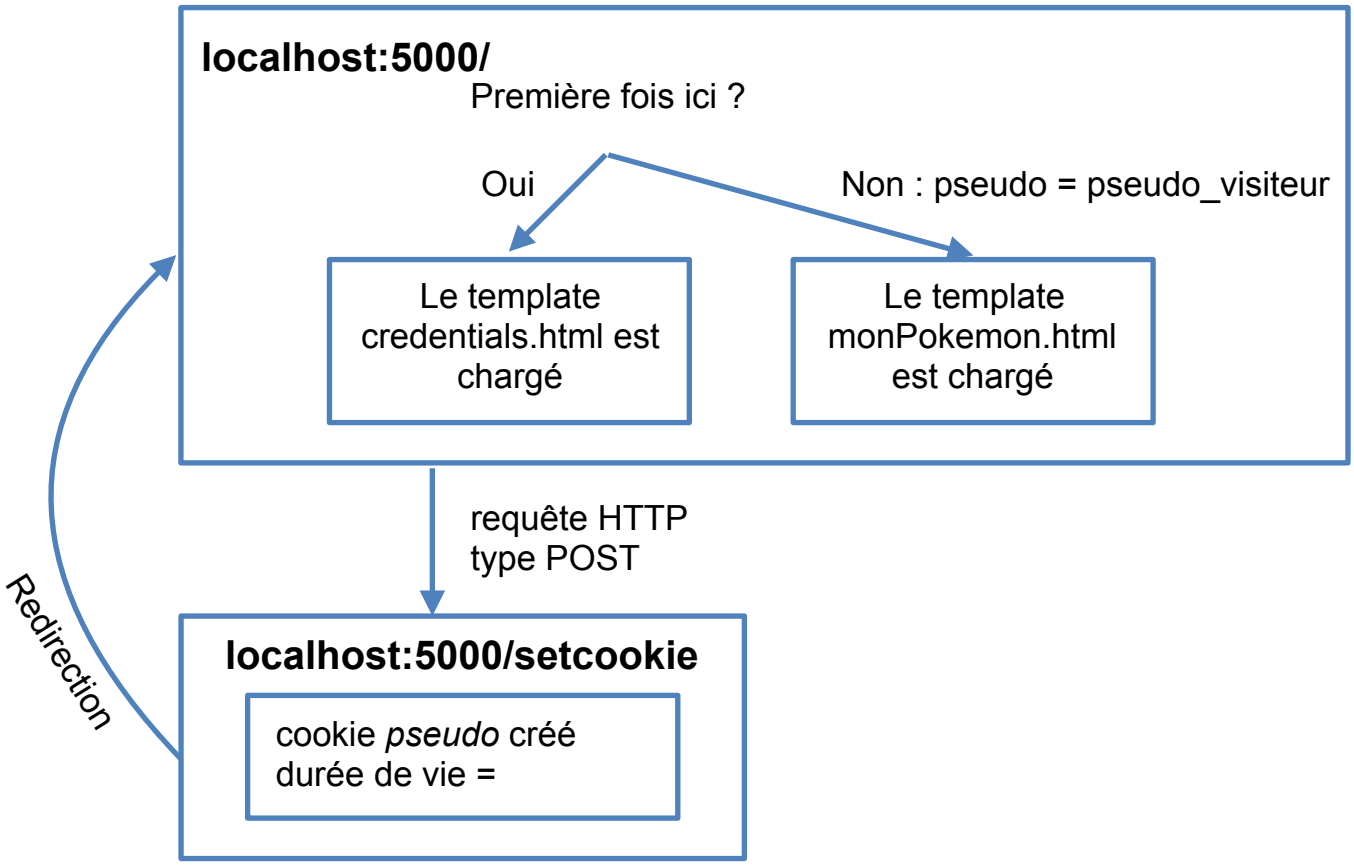
- Remplacez la ligne 12 par :
  <h1> Bienvenue {{ pseudo }}, créez le pokémon de vos rêves! </h1>

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

_____

_____

Reprenons la structure de notre site web pour comprendre :



❐ — À faire vous-même **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 pour chaque étape, redirection, requête quelle instruction est utilisée dans la fonction rerouting().

❖ 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 limité. Certains cookies peuvent être paramétrés pour avoir une durée de vie illimité.

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.

❐ — À faire vous-même **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 plutot 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…

❐ — À faire vous-même **9** —

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