# INF344 – Données du Web
# TP : *Tous les chemins mènent à la philosophie*

Antoine Amarilli

11 mai 2020

Ce TP vous propose d'écrire une application Web pour jouer à un jeu simple : en partant d'un article Wikipédia, atteindre l'article « Philosophie » en suivant les liens d'un article à l'autre, et en aussi peu d'étapes que possible. L'application du TP permet de jouer à ce jeu en récupérant les pages de Wikipédia et en proposant les liens à l'utilisateur.

Le TP utilise le langage de programmation Python et le framework Web Flask. Le squelette à compléter est sur le Moodle de INF344 `https://moodle.r2.enst.fr/moodle/course/view.php?id=87`. L'évaluation se compose de deux parties :

— Des tests automatiques, pour les Sections 2 et 4. L'évaluation de ces parties est effectuée par Moodle.
— Des tests manuels : est-ce que l'application est jouable, est-ce que le code HTML et CSS est propre et valide, est-ce que le design est réussi. Cette évaluation est effectuée par l'enseignant.

Le TP doit donc être rendu sur le Moodle sous la forme de deux fichiers :

— Le fichier `getpage.py`, pour l'évaluation automatisée.
— Le fichier `rendu.zip`, un fichier ZIP contenant tous les fichiers de votre projet, *y compris* `getpage.py`, pour évaluation manuelle. Attention à bien vérifier que votre rendu fonctionne et que tous les fichiers que vous utilisez sont fournis.

La date limite de rendu est le **18 mai 2020 à 13:30, heure de Paris**.

Consignes pour l'évaluation automatisée : une fois votre programme rendu sur Moodle, vous devez l'évaluer pour connaître votre note pour les tests automatisés. Pour ce faire, dans *Submission view* sur Moodle, cliquer sur Evaluate. L'évaluation prend environ une minute, et nécessite d'accepter temporairement un certificat SSL auto-signé (ceci ne fonctionne pas toujours sur Firefox). Le résultat de l'évaluation est une note sur 20 (sans indications de ce qui a fonctionné ou non). Vous pouvez également évaluer votre soumission contre un jeu de tests public (où des détails sur les erreurs seront fournis) : localement en lançant le fichier `tests.py` dans le squelette à compléter, ou sur Moodle en cliquant sur le bouton *Run* dans l'onglet *Edit* de Moodle (normalement ces deux méthodes devraient donner le même résultat).

Rappel : votre travail doit être **strictement personnel**, l'échange de code entre participants n'est pas autorisé, et vous ne devez pas mettre votre code en ligne. Tout plagiat entre rendus sera sanctionné par la note de 0 pour **tous** les rendus concernés. Les consignes précises sur le plagiat sont indiquées sur le Moodle.

## 1 Mise en place

Télécharger l'archive du squelette sur le Moodle et la décompresser. Dans le répertoire `philosophie/`, lancer `./getpage.py` et vérifier que « Ça fonctionne ! » s'affiche correctement. Installer les éventuelles librairies manquantes.

## 2 Interrogation de l'API Wikipédia

L'application Web affichera à l'utilisateur, lorsqu'il arrive sur un article de Wikipédia, la liste des 10 premiers liens de la page où il est arrivé. Cette information sera obtenue en interrogeant l'API de Wikipédia. Spécifiquement, nous allons éditer le fichier `getpage.py` et y programmer deux fonctions :

— Une fonction `getRawPage` qui prend en argument un titre de page `page`, et renvoie un couple formé de deux éléments : le nom de la page `page` après résolution des redirections, et le contenu HTML de la page. (La « résolution des redirections », signifie que le titre de la page « Philosophique » doit être « Philosophie » et non « Philosophique », parce qu'elle redirige vers « Philosophie ».)
— Une fonction `getPage` qui prend en argument un titre de page `page`, et renvoie un couple formé de deux éléments : le nom de la page `page` après résolution des redirections comme ci-dessus, et la liste des titres de page des 10 premiers liens de `page` (sans résolution des redirections pour ces titres).

**Attention :** comme cette partie est évaluée automatiquement, il faut respecter à la lettre les noms de fonction et leur spécification ci-dessus !

1. Nous allons effectuer une requête sur l'API Wikipédia pour recevoir un message JSON qui nous indique le contenu en HTML d'une page donnée de Wikipédia, en suivant les redirections.

   Compléter la fonction `getJSON` pour appeler l'URL `https://fr.wikipedia.org/w/api.php` avec les paramètres `GET` suivants : `format` vaut `json`, `action` vaut `parse`, `prop` vaut `text`, `redirects` vaut `true`, et `page` vaut le title de la page demandée.
2. Examiner la réponse JSON pour trouver le titre de la page après résolution des redirections, et son contenu HTML. Compléter la fonction `getRawPage` pour qu'elle récupère la réponse JSON avec `getJSON`, la lise avec `json.loads`, et retourne un couple qui consiste du titre de la page après redirection et de son contenu HTML.
3. Écrire une première version de la fonction `getPage(page)` qui récupère le titre et le contenu HTML de la page avec la fonction `getRawPage`, analyse le contenu HTML avec la bibliothèque BeautifulSoup, et renvoie un couple formé de deux éléments : le titre de la page après redirection, et la liste des valeurs de l'attribut `href` pour tous les liens du contenu HTML du document, dans l'ordre où les liens apparaissent. Si la page n'existe pas, la fonction devra renvoyer le couple `(None, [])`. On pourra s'inspirer des exemples de la documentation de BeautifulSoup. On utilisera le parseur `html.parser`.
4. Observer que `getPage` retourne beaucoup de résultats qui ne sont pas des titres de page corrects. Pour les filtrer, modifier la fonction `getPage` pour qu'elle ne renvoie que les liens apparaissant dans des éléments qui sont des descendants d'un élément `<p>` qui soit directement un enfant de l'élément `<div>` à la racine du contenu HTML. Ceci permettra d'éliminer les liens contenus dans les infoboxes, bannières, etc. (Indice : utiliser l'argument `recursive` de `find_all`.)
5. Modifier la fonction `getPage` pour éliminer les liens externes (liens vers des pages hors de Wikipédia), et les liens rouges (liens vers des pages inexistantes).

   À présent, la fonction `getPage` appliquée à la page `Utilisateur:A3nm/INF344` devrait renvoyer exactement les liens qui apparaissent après l'indication « Le bon contenu commence ici. », à part le lien rouge et le lien externe. On ne se souciera pas pour l'instant des problèmes de formatage sur ces liens (caractères accentués encodés, fragment, etc.) ; ils seront corrigés à la Section 4.
6. Modifier la fonction `getPage` pour retirer le préfixe `/wiki/` des liens.
7. Modifier la fonction `getPage` pour qu'elle renvoie les 10 premiers liens au plus.

# 3 Développement de l'application Web

Dans cette section, on s'intéresse au développement de l'application Web permettant de jouer au jeu, à l'aide de la fonction `getPage` réalisée à la section précédente.

1. Exécuter `./philosophie.py` dans un terminal. Ouvrir un navigateur Web sur `http://localhost:5000/`, et vérifier que le message « Bonjour, monde ! » s'affiche. Étudier le contenu du fichier `philosophie.py` et en comprendre le fonctionnement à l'aide du guide de démarrage rapide de Flask.

   Étudier également le fichier gabarit `templates/index.html` qui utilise le langage de gabarits Jinja.
2. Remplacer le contenu du bloc `body` du gabarit `index.html` par un formulaire qui contient un champ texte avec `name="start"` qui permet à l'utilisateur d'indiquer d'où il souhaite commencer la partie. La validation du formulaire devra déclencher une requête `POST` sur le chemin `/new-game` (pour lequel une route sera définie dans la prochaine question).
3. Ajouter une route `/new-game` au fichier `philosophie.py` pour la méthode `POST`, qui effectue les opérations suivantes : récupérer le titre de la page de départ fourni par l'utilisateur en utilisant `request.form`, le sauvegarder dans un champ `article` de l'objet `session`, et rediriger vers `/game`. On pourra se reporter aux sections "The Request Object" et "Sessions" du guide de démarrage rapide Flask.
4. Ajouter une route `game` qui effectue le rendu du nouveau gabarit `game.html` suivant :

   ```
   {% extends "index.html" %}

   {% block body %}
     {{ session.article }}
   {% endblock %}
   ```

   Vérifier que la soumission du formulaire fonctionne comme prévu. Noter qu'on a utilisé l'idiome de programmation Web *Post-redirect-get* pour éviter les soumissions multiples du formulaire.

5. Modifier la route `/game` pour appeler la fonction `getPage` de la section précédente, et passer le titre de la page courante (après redirection) et la liste de ses liens comme paramètres au gabarit `game.html`. Modifier le gabarit `game.html` pour afficher ces informations.
6. Modifier `game.html` pour que chaque titre de page liée soit rendu comme un bouton radio HTML, à l'intérieur d'un formulaire qui effectue une requête `POST` sur `/move` à la soumission. Chaque bouton radio devra avoir un élément `<label>` qui lui corresponde (en particulier, cliquer dessus doit activer le bouton), doit avoir `name="destination"`, et avoir avoir comme `value` le titre de la page cible.
7. Ajouter une route `/move` pour la méthode `POST` qui récupère la page où l'utilisateur souhaite se rendre, la sauvegarde dans l'objet de session, et redirige vers `/game`. Vérifier que l'on peut maintenant suivre un chemin sur Wikipédia en se déplaçant d'un article à l'autre.
8. Modifier `philosophie.py` pour tester si l'utilisateur a atteint la page « Philosophie ». Le cas échéant, rediriger vers la page d'index, et utiliser `flash` pour afficher une notification indiquant que la partie est gagnée. On se reportera pour ce faire à la section "Message flashing" de la documentation de Flask.
9. Ajouter un champ `score` à l'objet `session` qui est initialisé à 0 au début de la partie, est incrémenté à chaque déplacement, et est affiché dans `game.html` et dans le message de victoire à la fin de la partie.

# 4 Amélioration du traitement des liens

Cette section à améliorer la fonction `getPage` pour corriger les problèmes restants.

1. Pour limiter le nombre de requêtes envoyées à l'API Wikipédia, ajouter une variable globale `cache`, et s'en servir pour mémoïser les résultats de la fonction `getPage`, de la façon suivante : lorsque l'on invoque `getPage("Toto")` au cours de l'exécution du programme, la valeur de retour de cet appel doit être stocké dans `cache`, et les invocations subséquentes de `getPage("Toto")` doivent lire leur résultat dans `cache` sans faire d'appel à l'API.

   Vérifier que le cache est correctement utilisé lorsque l'utilisateur atteint une page qu'il avait déjà visitée précédemment. On ne cherchera pas à rendre ce cache persistant d'une exécution du programme à l'autre ; il s'agit seulement de mémoïser les résultats au cours d'une seule exécution.
2. Modifier `getPage` pour décoder les caractères non-ASCII des titres de page et des liens.
3. Modifier `getPage` pour retirer le fragment des liens de page quand il y en a un, par exemple pour retirer « #Frise_chronologique » de « Philosophie#Frise_chronologique ». Si le lien contenait uniquement un fragment et est rendu vide par cette opération, il doit être éliminé.
4. Modifier `getPage` pour que les sous-tirets dans les liens soient remplacés par des espaces.
5. Modifier `getPage` pour ne pas prendre en compte les liens vers des pages hors de l'espace de noms principal de Wikipédia, par exemple vers « Aide:Référence nécessaire ».
6. Modifier `getPage` pour que la sortie ne contienne pas de doublons. On fera en sorte que l'ordre des liens dans la liste reste celui de leur apparition dans la page, et on s'assurera que la fonction retourne toujours autant de liens différents que possible, dans la limite de dix.

# 5 Améliorations sur l'application Web

On corrige ici certains problèmes restants dans l'application Web.

1. Que se passe-t-il si, au cours d'une partie, on accède au jeu avec plusieurs onglets différents et on cherche à jouer alternativement dans les différents onglets ? Résoudre ce problème de sorte que, si l'utilisateur cherche à jouer dans un onglet alors que la partie a déjà progressé dans un autre onglet, alors le déplacement soit ignoré. On pourra utiliser un champ caché (`<input type="hidden">`) dans le formulaire.
2. Si l'utilisateur arrive sur une page où `getPage` n'extrait aucun lien (par exemple « Geoffrey Miller »), on souhaite le rediriger vers la page d'accueil en lui indiquant qu'il a perdu. Modifier le code en conséquence.
3. Modifier le code pour afficher un message si la page de départ demandée n'existe pas, si elle n'a pas de liens (comme à la question précédente), ou si c'est la page « Philosophie » ou une page qui redirige vers celle-ci.
4. Modifier le code pour que le premier bouton radio du formulaire soit présélectionné au changement.
5. Dans la version actuelle du code, l'utilisateur peut tricher en soumettant manuellement une requête POST qui ne correspond pas à un déplacement possible. Corriger le problème.

# 6 Mise en forme CSS

On cherchera dans cette section à rendre l'application plus esthétique à l'aide de CSS, et en modifiant les gabarits HTML. Les choix de design sont libres, dans la limite du bon goût. Quelques idées :

- Les pages devraient avoir un titre (`<h1>`), et le score et l'indication de la page courante dans un en-tête.
- Le titre des pages devrait permettre d'abandonner la partie en cours et de revenir à la page d'accueil.
- Les formulaires pourraient être centrés avec une largeur maximale fixée et une bordure.
- Les boutons de soumission devraient être à droite des formulaires.
- Les puces des listes de boutons radio devraient être enlevées, et de l'espacement ajouté entre les boutons.
- L'étiquette des boutons radio pourrait changer légèrement de couleur quand elle est survolée par la souris. Pour ce faire, il vaut mieux leur donner une largeur qui est celle du formulaire qui les contient.
- Les messages affichés à l'utilisateur pourraient être de couleurs différentes, par exemple vert quand l'utilisateur a gagné et rouge quand il s'agit d'erreurs.