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

Antoine Amarilli
a3nm@a3nm.net

3 mai 2017

Ce TP vous propose d'écrire une application Web simple pour jouer au jeu *Tous les chemins mènent à la philosophie*. Le but du jeu est de partir d'un article donné sur Wikipédia et d'atteindre l'article « Philosophie » en aussi peu d'étapes que possible, en suivant les liens internes menant d'un article à l'autre. L'application Web permet de jouer à ce jeu en récupérant les pages de Wikipédia et en proposant les dix premiers liens à l'utilisateur.

Le TP vous demande de compléter un squelette qui utilise le langage de programmation Python et le framework Web Flask. Le squelette, et les pointeurs vers la documentation pertinente, sont indiqués sur la page du cours `https://a3nm.net/work/teaching/#y2016-inf344`.

Votre travail sera évalué en fonction de la propreté du code Python, de la correction des résultats qu'il fournit (Sections 2 et 4, sur la base de tests automatisés), de son utilisabilité et de la validité du HTML et du CSS (Sections 3 et 5), et des qualités esthétiques du design (Section 6).

La durée du TP est de 3 heures. Vous pouvez continuer à travailler sur le TP par la suite si vous le souhaitez. Le TP doit être rendu par courrier électronique à l'adresse `inf344@a3nm.net` sous la forme d'un fichier ZIP contentant tous les fichiers que vous avez édités, au plus tard le **10 mai 2017 à dix-huit heures, heure de Paris**. En cas de rendu tardif (y compris si le rendu initial est corrompu, illisible, sans pièce jointe, etc.), un point sera déduit par heure de retard. Votre travail doit être **strictement personnel** : la discussion, l'échange de code entre participants n'est pas autorisé. Tout plagiat entre rendus sera sanctionné par la note de 0 pour tous les rendus concernés.

1 Mise en place

Télécharger l'archive du squelette sur la page du cours, et décompresser l'archive dans le répertoire personnel. Ouvrir un terminal dans le répertoire `philosophie/` (utiliser la commande `cd philosophie`) et essayer d'exécuter le programme `getpage.py` en tapant `./getpage.py` dans le terminal. Vérifier que « Ça marche ! » s'affiche correctement. Ouvrir le fichier `getpage.py` dans un éditeur de texte.

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 une fonction `getPage` qui retourne, étant donné un titre de page `page`, un couple qui consiste du nom de la page `page` après résolution des redirections, et de la liste des titres de page des 10 premiers liens de `page`. (Par « résolution des redirections », on entend que le titre de la page « Philosophique », qui est une redirection vers la page « Philosophie », doit être « Philosophie » et non « Philosophique ».)

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.

   En modifiant la fin du fichier, tester votre fonction `getJSON` et vérifier qu'elle fonctionne correctement. Ne pas hésiter à utiliser un navigateur Web pour tester manuellement des appels à l'API.

2. Examiner la réponse JSON pour trouver le titre de la page après résolution des redirections, et son texte HTML. Si nécessaire, on pourra utiliser Python pour embellir (*pretty-print*) le JSON. Compléter en conséquence 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 consistant du titre de la page après redirection et de 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 `<p>` directement à la racine : ceci permettra d'éliminer les liens contenus dans les infoboxes, bannières, etc. (Indice : utiliser l'argument `recursive` de `find_all`.)
5. Trouver une manière de filtrer les liens pour éliminer les liens externes (liens vers des pages hors de Wikipédia), et les liens rouges (liens vers des pages inexistantes). Modifier la fonction `getPage` en conséquence.

   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 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. On pourra utiliser le *slicing* en Python, c'est-à-dire la notation `s[i:j]` pour retourner les caractères de i (inclus) à j (exclu) de la chaîne s ; cette notation fonctionne également pour les listes.
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 « Tous les chemins mènent à la philosophie », à l'aide de la fonction `getPage` réalisée à la section précédente.

1. Exécuter le fichier `philosophie.py` du squelette dans un terminal. Ouvrir un navigateur Web, le pointer sur l'URL `http://localhost:5000/`, et vérifier que le message « Bonjour, monde ! » s'affiche correctement. 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`. Dans le langage Jinja, la directive `{% block body %}` définit un *bloc* nommé `body`, qui peut être ignoré pour l'instant. Les lignes de la forme `{# ... #}` sont des commentaires.
2. Remplacer le contenu du bloc `body` du gabarit `index.html` par un formulaire qui effectue une requête `POST` sur `/new-game` lorsqu'on le valide, et qui contient un champ texte permettant à l'utilisateur d'indiquer le titre de la page d'où ils souhaitent commencer la partie. Utiliser `placeholder` ou `<label>` pour que le rôle de ce champ soit clair pour l'utilisateur. Dans `philosophie.py`, modifier la fonction `index` pour retirer le paramètre `message` devenu inutile.

   Rafraîchir le navigateur et vérifier que le rendu du formulaire est correct.
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. On ne cherchera pas à utiliser `url_for`. On remarquera que l'objet `session` sauvegarde ses données dans des cookies qui sont signées afin que l'utilisateur ne puisse pas les modifier.

   On remarquera également que Flask, lorsqu'il fonctionne en mode de débogage, fournit des traces d'appel du code Python directement dans le navigateur en cas de problème à l'exécution. Par ailleurs, lorsque le fichier source est modifié, il recharge automatiquement le nouveau fichier, de sorte qu'il n'est pas nécessaire de relancer Flask. En revanche, lorsque le fichier source contient une erreur de syntaxe, Flask quitte, et doit être relancé une fois l'erreur corrigée.

4. Ajouter une route `game` pour la méthode `GET`, qui effectue le rendu d'un nouveau gabarit `game.html` qu'on définira comme suit :

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

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

   Observer que la directive Jinja `extends` permet à `game.html` de réutiliser le code de `index.html` excepté le bloc `body`, qui est modifié. Ceci permet d'éviter la répétition des incantations propitiatoires de HTML.

   Vérifier que l'on peut correctement soumettre le formulaire et que la valeur est correctement affichée sur la page de résultats.

   Noter que les routes `/new-game` et `game/` utilisent 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 ; on utilisera notamment la fonctionnalité suivante de Jinja qui permet d'itérer sur une collection :

```
{% for item in items %}
  <li>{{ item }}</li>
{% endfor %}
```

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. On pourra utiliser la variable `loop.index` de Jinja pour générer des identifiants uniques aux boutons radio, afin de leur attribuer des étiquettes (éléments HTML `<label>`) permettant de connaître le titre du lien. Vérifier que le formulaire est correctement rendu et que les `<label>` sont correctement appariés aux boutons radio lorsque l'on clique dessus.

   On rappelle que l'attribut `id` dans un document HTML valide doit commencer par une lettre et non par un chiffre.

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, et on modifiera `index.html` pour ajouter le code nécessaire dans `<body>` (mais hors du bloc `body`).

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

On observe que beaucoup de liens ne sont pas correctement gérés. Cette section vise à régler les problèmes restants dans `getpage.py`.

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 Wikipédia.

   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. On pourra utiliser la fonction `unquote` de `urllib.parse`.

3. Modifier `getPage` pour supprimer le fragment des titres de page, par exemple pour retirer le fragment « #Frise_chronologique » de « Philosophie#Frise_chronologique ». On pourra utiliser la fonction `urldefrag` de `urllib.parse`.
4. Modifier `getPage` pour que les sous-tirets dans les titres de page 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 titres 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 ?

   Pour corriger ce problème, ajouter à `game.html` un champ caché contenant le score actuel, et modifier le code de `/move` pour vérifier que le score dans le formulaire soumis est toujours égal au score stocké dans l'objet de session. Si ce n'est pas le cas, cela signifie que l'utilisateur a déjà progressé dans un autre onglet et que le déplacement doit être ignoré : on en avertira l'utilisateur avec `flash`, sauf s'il se trouve que le déplacement demandé mène l'utilisateur à l'article où il se trouve actuellement.

   Vérifier que cette modification corrige effectivement le problème.
2. Que se passe-t-il quand l'utilisateur arrive sur une page où `getPage` n'extrait aucun lien ? (Par exemple « Geoffrey Miller ».) Corriger le programme pour rediriger l'utilisateur vers la page d'accueil dans ce cas, en affichant un message avec `flash`.
3. Modifier le code pour afficher un message adapté à l'utilisateur si la page de départ demandée n'existe pas. Afficher également un message adaptée si elle n'a pas de liens (comme à la question précédente).

   Vérifier que votre programme réagit correctement si l'utilisateur demande à commencer à la page « Philosophie », ou à une page qui redirige vers celle-ci, par exemple « Philosophique ».
4. Que se passe-t-il quand l'utilisateur soumet le formulaire de déplacement sans avoir sélectionné de bouton radio ? Corriger le problème en présélectionnant le premier bouton radio à l'aide de la variable Jinja `loop.first`.
5. Utiliser les outils de développement de votre navigateur pour modifier les options du formulaire en cours de partie et tricher en atteignant immédiatement « Philosophie ». Empêcher cela en modifiant le code de la route `/move` pour vérifier que le déplacement demandé est effectivement possible, et pour refuser le déplacement avec un message dans le cas contraire.

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. On pourra en particulier se concentrer sur les points suivants :

— Les pages devraient avoir un titre (`<h1>`) qui peut être rendu avec le score et l'indication de la page courante dans un en-tête, en blanc sur un arrière-plan foncé.
— Le titre des pages devrait permettre d'abandonner la partie en cours et de revenir à la page d'accueil, mais il est préférable de ne pas le décorer comme un lien.
— Les formulaires pourraient être centrés avec une largeur maximale fixée et une bordure `outset`.
— 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. On pourra notamment chercher à utiliser l'invocation `get_flashed_messages(with_categories=True)`.