TP - Web scraping pour construire une carte interactive en HTML

Alexander Delaporte

Web scraping pour construire une carte interactive en HTML

Dans cet exercice, vous allez produire une carte interactive en HTML à partir de données extraites d'une page web. Pour ce faire, vous allez :

1. Extraire les données petinentes d'un tableau présenté sur une page web.
2. Les stocker dans un format de données approprié.
3. Déclarer et paramétrer une carte.
4. Créer une couche de données pour y ajouter les points dont vous avez obtenu les coordonnées.

Récupérer les coordonnées géographiques des points

Vous trouverez sur cette page un tableau de données correspondant aux langues de la famille Tai-Kadai. Chaque ligne du tableau représente une langue, qui sera représentée par un point sur la carte.

Imports

Vous n'avez pas besoin de modifier la cellule ci-dessous, elle contient les modules et librairies Python dont vous aurez besoin pour cette partie du travail. Pour que le programme fonctionne, n'oubliez pas de l'exécuter quand même ! (`ctrl` + `enter` ou `shift` + `enter`)

In [ ]:
```
# Parsing HTML
from bs4 import BeautifulSoup as soup
# Client web
from urllib.request import urlopen as uReq
import urllib.request
```

Fonctions

Vous pouvez également laisser telle quelle la cellule suivante.

Elle comporte 2 fonctions que je vous fournis pour faciliter votre travail :

1. `url_ok()` prend en paramètre une URL et vérifie si elle est accessible. Vous n'aurez pas besoin d'appeler directement cette fonction, elle sera utilisée par la fonction `good_soup()`.
2. `good_soup()` prend elle aussi une URL en paramètre. Si l'URL est accessible (= validée par `url_ok()`), cette fonction renvoie le HTML parsé que vous pourrez ensuite parcourir.

In [ ]:
```
def url_ok(url):
    """Teste si une URL est accessible, paramètre = URl (str)"""
    request = urllib.request.Request(url)
    request.get_method = lambda: 'HEAD'
```

```
    try:
        urllib.request.urlopen(request)
        return True
    except urllib.request.HTTPError:
        return False


def good_soup(url):
    """Parser le HTML d'une page web avec bs4, paramètre = URL (str)"""
    if url_ok(url):
        uClient = uReq(url)
        page_html = uClient.read()
        uClient.close()
        page_soup = soup(page_html, "html.parser")
        return page_soup
```

Et maintenant, à vous de jouer !

Extraire le tableau de la page

Afin d'explorer le code source d'une page web, il faut préalablement le parser pour pouvoir l'explorer automatiquement. C'est exactement ce que fait la fonction `good_soup()` présentée ci-dessus.

Appelez cette fonction sur l'URL de la page où se trouve le tableau, en associant le résultat à la variable `ma_soupe`.

In [ ]:
```
# Appeler la fonction good_soup() sur l'URL
```

S'il n'y a pas d'erreur, appeler `ma_soupe` ou utiliser `print(ma_soupe)` devrait afficher le code source de la page HTML.

In [ ]:
```
ma_soupe
```

La page ne contient pas que le tableau, il va falloir l'isoler du reste du contenu.

Pour récupérer un élément unique sur une page, la librairie `bs4` (documentation) dispose d'une fonction `find()`. Elle peut par exemple prendre en argument :

- L'identifiant de l'élément avec la syntaxe `objetbs4.find(id="identifiant_element")`
- Le tag de l'élément avec la syntaxe `objetbs4.find("tag")`

Dans le second cas, si plusieurs éléments de la page comportent le même tag, seul le premier sera récupéré.

Ici, il n'y a qu'un tableau sur la page et celui-ci porte un identifiant, les deux solutions mèneront au même résultat. Le contenu de votre variable `ma_soupe` étant un objet `bs4`, vous pouvez appeler la fonction `find()` dessus.

Dans la cellule ci-dessous, appelez `find()` sur `ma_soupe` en renseignant l'argument de façon à obtenir le code HTML correspondant uniquement au tableau. Le résultat sera associé à la variable `tableau`.

In [ ]:
```
# isoler le tableau dans le code source de la page (ma_soupe)
```

Comme pour le parsing de la page dans son ensemble, vous pouvez vérifier qu'il n'y a pas de problème :

In [ ]:
```
# vérifier le contenu de la variable tableau
```

Filtrer le contenu du tableau

Pour produire une carte, les seules données indispensables sont les coordonnées géographiques de chaque point. Il est nécessaire de les récupérer.

Par ailleurs, pour que les données restent interprétables, il serait aussi préférable de conserver une indication de ce à quoi correspond chaque paire de coordonnées, comme le nom et/ou l'identifiant de la langue.

Pour parcourir les lignes du tableau, il n'est pas possible d'utiliser `find("tr")` qui ne renverrait que la première ligne. Heureusement, la fonction `find_all()` s'utilise de façon similaire et renvoie tous les éléments correspondant à la requête. Le résultat de `find_all()` peut être parcouru avec une boucle `for`.

Complétez la cellule ci-dessous pour afficher chacune des lignes du tableau :

In [ ]:
```
for ligne in XXX:
    print(ligne)
```

La ligne d'étiquettes n'est pas une ligne de données, nous ne voulons pas la récupérer ! Si ce n'est pas fait, adaptez le code de la cellule précédente pour n'afficher que les lignes de données.

Vous l'aviez peut-être deviné en appliquant la consigne précédente, mais `find_all()` permet d'appliquer un index à ses résultats. En reprenant la boucle `for` qui parcourt les lignes du tableau, vous pouvez maintenant afficher uniquement les colonnes pertinentes du tableau.

Note : l'index débute à 0 et non à 1.

In [ ]:
```
# n'afficher que les lignes correspondant à la longitude, la latitude, le nom et l'ident
```

Très bien, les données que nous cherchons à récupérer sont bien là, mais elles sont encore entourées de balises `<td>`. Pour les supprimer, vous pouvez faire appel à l'attribut `.text` des objets `bs4`, qui correspond au contenu textuel de l'élément concerné. Par exemple :

In [ ]:
```
ma_soupe.find("h1").text
```

Stocker les données dans un dictionnaire

Modifier la boucle `for` en appelant l'attribut `.text` des éléments permettrait d'afficher le texte dans la console, mais en l'état les données ne seront enregistrées nulle part et il ne sera donc pas possible de les réutiliser sans devoir les produire à nouveau.

La solution qui sera utilisée ici est la suivante :

- L'ensemble des lignes sera stockée dans une liste.
- Dans cette liste, chacune des lignes correspondra à un dictionnaire, c'est-à-dire une liste associative.

Une liste vide est initiée avec `[]`.

Un dictionnaire vide s'initie avec `{}`. Pour déclarer un nouveau dictionnaire en ajoutant directement les données, la syntaxe est la suivante :

```
nom_dictionnaire = {"clé1": valeur1, "clé2": valeur2, ..., "cléX": valeurX}
```

Pour ajouter des données à un dictionnaire déjà créé, vous pouvez utiliser :

```
nom_dictionnaire["clé"]= valeur
```

Voici déjà la liste qui va accueillir l'ensemble des données.

In [ ]:
```
# Liste qui va accueillir les dictionnaires
donnees = []
```

Il faut maintenant peupler cette liste, en lui ajoutant un dictionnaire pour chaque ligne extraite du tableau.

En vous aidant de la boucle `for` déjà écrite et de la documentation, complétez la fonction ci-dessous pour obtenir la liste des données :

In [ ]:
```
for ligne in XXX:
    dico_ligne = {
        # longitude
        # latitude
        # nom
        # identifiant
    }
    # ajouter dico_ligne à la liste donnees
```

S'il n'y a pas d'erreur, le début du contenu de `donnees` devrait ressembler à ceci :

```
[{'longitude': '104.812',
  'latitude': '23.2384',
  'nom': 'Pubiao-Qabiao',
  'id': 'qabi1235'},
 {'longitude': '105.754',
  'latitude': '23.342',
  'nom': 'Yerong-Southern Buyang',
  'id': 'yero1238'},
```

Appelez la liste `donnees` pour vérifier qu'il n'y a pas de problème.

In [ ]:
```
# vérifier le contenu de donnees
```

Construire la représentation cartographique

Import de la librairie `folium`

Pour produire une carte au format HTML, vous aurez besoin de la librairie `folium`. Celle-ci est importée dans la cellule ci-dessous, qui n'a pas besoin d'être modifiée.

In [ ]:
```
import folium
```

Mettre en place le fond de carte

Avant de représenter les données, il faut d'abord déclarer et paramétrer la carte, via une instance d'objet `Map` de la librairie `folium`. Vous pouvez éventuellement initialiser l'objet sans lui attribuer de paramètre, en l'associant à la variable `ma_carte`.

In [ ]:
```
# déclarez la variable ma_carte en lui associant une instance d'objet folium Map() sans
```

Pour obtenir un aperçu de la carte, il suffit d'appeler la variable dans laquelle elle est stockée.

In [ ]:
```
# prévisualiser la carte
```

Vous obtenez bien une carte, mais cette vue par défaut n'est ni satisfaisante ni optimale pour visualiser des données.

Déclarez à nouveau votre carte, en renseignant au moins les paramètres `location` (coordonnées par défaut), `zoom_start` (zoom par défaut, basé sur la valeur de `location`), et `tiles` (fond de carte par défaut).

Pour ce faire, vous pouvez vous aider de cette page de la documentation de `folium`. Si vous ne lisez pas l'anglais, cet article de *Tekipaki* peut également vous apporter des informations.

In [ ]:
```
# déclarer à nouveau la carte avec ses paramètres
```

In [ ]:
```
# prévisualiser à nouveau la carte
```

C'est (normalement) beaucoup mieux comme ça !

Déclarer une couche de données

Il est temps d'ajouter à la carte les données extraites du tableau HTML. Voici comment procéder :

1. Déclarer une couche de données
2. Ajouter les données à la couche
3. Ajouter la couche à la carte

La couche de données correspond à un objet de `folium` nommé `FeatureGroup`. Vous trouverez ses différents paramètres dans la documentation, mais vous pouvez ici l'initialiser avec seulement le paramètre `name`.

Pour pouvoir l'ajouter à la carte ensuite il est indispensable de stocker les informations de la couche de données dans une variable ; je vous conseille de la nommer de façon à identifier quelles sont les données correspondantes (vous pouvez par exemple reprendre l'identifiant Glottolog de la famille Tai-Kadai).

In [ ]:
```
# déclarer une couche de données en renseignant le paramètre name, l'associer à une vari
```

Reporter les coordonnées géographiques dans la couche de données

La couche de données existe maintenant dans le sens où une instance d'objet `FeatureGroup` a été créée, mais elle ne comporte aucune information (à part son nom), et n'est pas reliée à la carte.

Pour rappel, les coordonnées sont actuellement stockées dans la variable `donnees`, sous forme d'une liste de dictionnaires.

Celles-ci correspondent à une série de points. Dans le code, chaque point correspondra à un objet `folium.CircleMarker()` qui prendra en paramètre les coordonnées géographiques dans une liste `[latitude, longitude]`.

Vous disposez en fait déjà de tout ce qu'il vous faut pour ajouter vos points à la couche de données.\
Sachant qu'un point est ajouté avec la fonction `.add_to(nom_de_la_couche_de_donnees)`, décommentez et complétez le code de la cellule suivante :

In [ ]:
```
"""
for XXX in XXX:
    folium.CircleMarker(
        location=XXX,
    ).add_to(XXX)
"""
```

Ajouter la couche de données à la carte

Le plus dur est fait. La fonction `.add_to()` permet également d'ajouter une couche de données à une carte, en utilisant la syntaxe `nom_de_la_couche_de_donnees.add_to(nom_de_la_carte)`.

In [ ]:
```
# ajouter la couche de données à la carte
```

Félicitations, vous avez terminé ! 🎉

Les points apparaîtront maintenant lorsque vous visualiserez votre carte. Vous pouvez admirer le résultat de votre travail en appelant la variable correspondant à la carte.

In [ ]:
```
# visualiser la carte
```

Aller plus loin

Si vous voulez continuer à vous amuser avec les représentations cartographiques produites par `folium`, voici quelques suggestions sur ce que vous pouvez essayer :

- Paramétrer plus finement l'aspect des marqueurs
- Exporter la carte dans un fichier HTML
- Tracer d'autres séries de points ou diviser les points en plusieurs séries
- Tracer un ou plusieurs polygones
- Capturer l'image de la carte dans un fichier PNG
- Mettre en place un *layer control*
- Définir des fonctions pour répéter aisément les traitements

Les différents articles consacrés à la cartographie de données linguistiques sur *Tekipaki* pourront vous être utiles si besoin.

Code source disponible : <a xmlns:dct="http://purl.org/dc/terms/" href="https://github.com/alxdrdelaporte/LTTAC_2021_TP" rel="dct:source">https://github.com/alxdrdelaporte/LTTAC_2021_TP.