ProgRes - TP - Web Api

Exercice 1

1. Ecrire une API dont la route `/sum/<a:int>/<b:int>` qui affiche la somme des deux nombres passés en arguments
2. Ecrire une API dont la route `/title/<url:path>` accepte l'url d'un site web et retourne le titre de la page (utiliser la librairie Python html.parser)
3. Ajouter une route `/content/<url:path>` qui accepte l'url d'une actualité du site nextinpact.com et retourne uniquement le titre et le contenu de l'actualité.
4. Tester la route `/content/<url:path>` avec un article du site jeuxvideo.com et avec un article du site lemonde.fr. Corriger les erreurs éventuelles.
5. ⭐ Ajouter une route `/styled-content/<url:path>` qui accepte l'url d'un l'article d'un site d'actualité et retourne uniquement l'article (le titre et le contenu) avec un style standardisé (le même quelque soit le site d'actualité) pour une *lecture agréable*.

Exercice 2

1. Télécharger les données publique (**au format CSV**) concernant les arbres d'alignement de Paris sur le site de l'opendata de Paris. Créer une fonction qui parse le fichier csv dans une liste de dictionnaires associant le nom des champs à leur valeur (on pourra ignorer le champs adresse). Soit dans le format suivant :

```
[{'geopoint': .., 'circonference': .., 'hauteur': .., ..}, { ..}, ..]
```

1. Crée une API avec une route `/licence` qui affiche la licence d'utilisation de l'ensemble de données précédent.
2. Ajouter une route `/arbres/<id:int>` qui affiche les informations concernant l'arbre d'identifiant `id`.
3. Ajouter une route `/search/arbres` qui accepte un paramètre GET `query` de la forme `circonference:VALEUR` et retourne les 100 premiers arbres ayant la circonference demandée.
4. Permettre au paramètre `query` d'être de la forme `hauteur:VALEUR` ou `date:VALEUR`.
5. Permettre au paramètre `query` d'être de la forme `espece:VALEUR` affichant les 100 premiers arbres dont l'espèce *contient* le texte `VALEUR`.
6. Crée un nouveau programme python qui permet de tester chaque route de votre API, avec différents paramètres, en utilisant le package `requests`.
7. Permettre au paramètre `query` d'être de la forme `geopoint:VALEUR` affichant l'arbre le plus près de la position demandée.
8. Permettre au paramètre `query` d'être de la forme `in-rectangle:VALEUR,VALEUR` affichant les arbres dont la position est dans le rectangles délimité par les deux valeurs passées en paramètre.
9. ⭐ Pour les `query` concernant des nombres, autoriser la recherche avec conditions, par exemple `query=champs:<=VALEUR`.
10. ⭐ Permettre la recherche sur plusieurs critères au format `query=champs1:valeur1,champs2:valeurs2`
11. ⭐🐒🌳🌳🌳🍌 Ajouter une route `/chemin/<origin>/<destination>` acceptant le paramètre GET step, qui retourne le plus cours chemin pour aller du point `origin` au point `destination` en passant d'arbre en arbre, en autorisant une distance d'au plus `step` entre deux arbres consécutifs (et entre le point d'origin et le premier arbre, et le point de destination et le dernier arbre).