UNIVERSITÉ DE CAEN **Année universitaire 2024-2025**

IUT Grand Ouest Normandie **Département Réseaux et Télécommunications**

SAÉ 1.05 — Traiter des données — Aide au démarrage du projet

Martin Pépin

Ce document est un accompagnement au projet. Profitez des heures de TP pour avancer au maximum dessus avec l'aide des enseignant·es. Vous allez d'abord découvrir l'API REST du projet PokeAPI et comment interagir avec depuis un programme Python. Vous verrez ensuite comment écrire un document dans le langage Markdown et comment le convertir en HTML.

1 Prérequis techniques

On a besoin d'installer des bibliothèques utiles pour le projet. On va pour cela utiliser le programme **pip** (*Python Install Package*) ou **pipx** selon les salles de TP. Dans un terminal, taper la commande :

- en 3147 ou sur vos machines personnelles :
  `pip3 install --user --upgrade markdown requests` ;
- dans les salles de TP réseau/télécom : `pipx install markdown requests`

Si tout a bien fonctionné, dans l'interpréteur Python vous devriez pouvoir importer ces modules sans que cela génère de message d'erreur.

```
import markdown
import requests
```

NB : si jamais vous faites le TP sur Windows, l'installeur pip est utilisable en ouvrant un *cmd* et en tapant `python -m pip install requests markdown` (par exemple)

2 PokeAPI

Exercice 1. Découverte de PokeAPI avec le navigateur

PokeAPI est une base de données répertoriant des informations détaillées sur le jeu Pokemon et ses différentes générations. Le site web de PokeAPI est disponible à l'url suivante : https://pokeapi.co/ et l'API (pour les ordinateurs) est à l'adresse https://pokeapi.co/api/v2/. "https ://raw.githubuserco…es/pokemon/latest/25.ogg" Voyons ça de plus près :

1. Rendez vous sur la documentation de l'API et cherchez le point d'API permettant de télécharger des **lieux**. À l'aide du navigateur, accédez au lieu d'identifiant 160.
2. Cherchez dans la documentation comment accéder à des Pokemons. Cherchez le Pokémon d'identifiant 25. Cherchez les réponses aux questions suivantes à l'aide de la réponse de l'API ainsi que la documentation quand c'est nécessaire.

(a) Quel est son type ?

(b) Quel est son poids ?

(c) Combien de « *moves* » ce Pokémon a-t-il ?

(d) D'après la doc, que son les « *moves* » ? Comment diriez vous en français ?

(e) Déroulez les premires « *moves* » et examinez les. En quelle langue son écrits leurs noms ?

(f) Une URL est fournie avec chaque « *move* », cherchez la documentation de cette URL dans la doc.

(g) À l'aide de la documentation et de la bonne requête, trouver le noms de quelques « *moves* » en français (fr) et en espagnol (es).

Exercice 2. Test de l'API avec `requests`

On se propose maintenant d'interagir directement en Python avec l'API, en utilisant la très bonne bibliothèque `requests` pour faire de requêtes HTTP. Lien vers la documentation de la bibliothèque : https://docs.python-requests.org/en/latest/.

1. Dans l'interpréteur Python : utiliser la librairie *requests* effectuer la bonne requête sur l'API OSM pour récupérer les informations sur le Pokemon d'identifiant 25. Stocker le résultat dans une variable `response`.
   Si vous avez fait les choses correctement, vous devriez pouvoir effectuer les suivantes :

   ```
   >>> response = requests.get("URL_À_ENTRER_VOUS_MÊME")
   >>> help(response)
   ???
   >>> reponse.status_code
   ???
   >>> response.text  # tout au format brut, berk !
   ???
   ```

2. On va maintenant convertir le résultat au format JSON et l'exploiter. Tester :

   ```
   >>> json_data = response.json()
   >>> json_data
   ???
   ```

   Vous devriez voir un dictionnaire.
3. Exploitez la variable `json_data` pour afficher le nom du Pokémon, son poids, sa taille.
4. Que contient la liste stockée à la clef `"stats"` ?
5. Accédez au premier élément de cette liste et stockez le dans une variable `stat1`.
   (a) Affichez sa valeur de base (`"base_stat"`) dans l'interpréteur.
   (b) Affichez son nom (où est-il stocké ?) dans l'interpréteur.

6. Jusque là on n'a fait que travailler dans l'interpréteur, c'est bien pour tester et comprendre le fonctionnement. Maintenant, créer un fichier Python qui contiendra une fonction `download_poke`, prenant en argument un identifiant entier `id`, et qui renvoie les informations sur le Pokémon numéro `id`

   Exemple :

   ```
   >>> poke25 = download_poke(25)
   >>> poke25["name"]
   'pikachu'
   >>> poke42 = download_poke(42)
   >>> poke42["name"]
   'golbat'
   ```

7. Écrivez une fonction `print_poke_stats` prenant en argument les données sur un Pokémon et affichant de façon lisible les statistiques de base du Pokémon (stockées à la clef `"stats"`).

   ```
   >>> pika = download_poke(25)
   >>> print_poke_stats(pika)
   hp 35
   attack: 55
   defense: 40
   special-attack: 50
   special-defense: 50
   speed: 90
   ```

Exercice 3. Statistiques de base

1. D'après la documentation (rubrique « *Resource Lists/Pagination* ») que se passe-t-il quand on interroge un point d'API sans préciser d'identifiant ? Par exemple https://pokeapi.co/api/v2/pokemon/.
2. Combien d'éléments obtient-on par défaut ?
3. Comment faire pour télécharger les 50 premiers Pokémons de l'API ?
4. Calculer la moyenne des points de vie des 50 premiers Pokémons de l'API.
5. Comment faire pour télécharger tous les Pokémons ?

3 Markdown

Exercice 4. Conversion Markdown → HTML

1. Vous connaissez déjà le langage Markdown, utilisé pour les comptes rendus et notices de TP. Pour rappel, une documentation est disponible à l'url https://www.markdownguide.org/cheat-sheet/
2. On souhaite maintenant s'intéresser à la génération de fichiers HTML depuis un script Python. Écrire une fonction `ouput_list_md(my_list: list[str], filename: str)` prenant en argument une liste de chaînes de caractères et un nom de fichier, et implémentant le comportement suivant :

— Elle ouvre le fichier `filename` en écriture.
— Elle écrit dans le fichier les informations suivantes, dans cet ordre :
   (a) un titre de niveau 1, par exemple : « Test Markdown depuis Python »
   (b) un titre de niveau 2 qui dit « Une liste de XXX éléments » (XXX à remplacer par la longueur de `my_list`) ;
   (c) une liste Markdown non ordonnée qui contient tous les éléments de la liste.

3. Testez votre fonction avec une liste de 3/4 éléments, par exemple :

```
>>> convert_list(["Patate", "navet", "chou", "oignon"])
```

   Une fois votre fonction appelée, vérifiez qu'un fichier a bien été créé et qu'il contient du Markdown valide.

4. En vous aidant de la documentation (https://python-markdown.github.io/reference/#the-basics, utilisez le module Markdown pour convertir votre fichier Markdown en HTML. Il faudra
   (a) ouvrir le fichier Markdown en lecture depuis Python ;
   (b) utiliser la bonne fonction du module `markdown` (ne cherchez pas compliqué, c'est au début de la doc) ;
   (c) ouvrir un fichier HTML en écriture et y écrire le résultat.

4 Notions « avancées »

Exercice 5. Traductions

1. D'après la documentation du point d'API `Stat` (https://pokeapi.co/docs/v2#stat), à quoi sert l'attribut `names` ?
2. Modifier la fonction `print_poke_stats` de l'exercice précédent pour qu'elle affiche les statistiques en français. Pour cela, il faudra faire une nouvelle requête GET sur l'API.

Exercice 6. Implémenter un cache

Vous avez dû le constater, faire une requête sur l'API est lent. Cet exercice vous montre comment implémenter un cache très basique. Un cache est un mécanisme permettant d'éviter de télécharger plusieurs fois la même donnée.

1. Créez un dossier `cache` à côté de votre code.
2. Au début de votre fichier Python, importez les modules `json` et `os` à l'aides des instructions :

```
import json
import os
```

3. Écrivez une fonction `download(url: str) -> dict` qui prend une URL en paramètre et télécharge le fichier JSON stocké à cette URL.
4. Dans un interpréteur, exécutez les instructions suivantes (après avoir chargé/importé votre fonction `download`) :

```
>>> pika = download("https://pokeapi.co/api/v2/pokemon/25/")
>>> with open("cache/25.json", "w") as file
...     json.dump(pika, file)
```

Que contient désormais le fichier `cache/25.json` ?

On a sauvegardé le Pokémon au format JSON dans un fichier à l'aide de la fonction `json.dump` (documentation : https://docs.python.org/3.11/library/json.html#json.dump).

5. Dans un interpréteur, tapez désormais :

```
>>> with open("cache/25.json", "r") as file:
...     file_content = json.load(file)
>>> file_content["name"]
```

Qu'est ce qui s'affiche ?

On a lu le contenu du fichier JSON `cache/25.json` et on l'a transformé en dictionnaire Python (documentation : https://docs.python.org/3.11/library/json.html#json.load).

6. À l'aide de la documentation (https://docs.python.org/3/library/os.path.html#os.path.isfile), que fait la fonction `os.path.isfile` de la bibliothèque standard de Python ?

Testez la fonction à l'aide des instructions suivantes (dans l'interpréteur) :

```
>>> os.path.isfile("cache/25.json")
???
>>> os.path.isfile("cache/999999999.json")
???
```

7. Écrivez une fonction `download_poke_cached(id: int) -> dict` qui implémente le comportement suivant :
   - à l'aide de `os.path.isfile`, on regarde si le dossier `cache/` contient un fichier nommé `f"{id}.json"` ;
   - si le fichier existe, on lit le fichier à l'aide de `json.load` et on renvoie le résultat ;
   - si le fichier n'existe pas
     - on télécharge les données du Pokémon `id` depuis la PokeAPI (réutiliser la fonction `download`),
     - on sauvegarde les données dans le fichier `f"cache/{id}.json"` à l'aide de `json.dump`,
     - on renvoie les données.

   Si vous avez correctement codé la fonction
   - la première fois que vous exécutez `download_poke_cached(77)`, vous téléchargez le Pokémon ponyta en environ une seconde et vous créez un fichier `77.json` dans le dossier cache ;
   - la deuxième fois que vous exécutez `download_poke_cached(77)`, la fonction vous répond instantanément en allant lire le fichier `cache/77.json`.

   Testez.