Traiter des données : cahier des charges

1 Généralités

1.1 Organisation

— Projet à réaliser en **binôme**. Vous devez renseigner vos groupes avant le Vendredi 17/12/22 sur Unicloud . S'il y a un nombre d'étudiants impair, il faudra que certains travaillent en autonomie (trinôme interdit) → `https://unicloud.unicaen.fr/index.php/s/FSXoxzefFT2pRCa` → "Ouvrir dans OnlyOffice"

— Une personne par binôme fera le rendu du travail sur Ecampus. Celui-ci sera votre code implémentant les fonctionnalités demandées dans le cahier des charges. Date de fermeture du dépôt : **Vendredi 14/01/22** à 23h59. Retard → pénalité.

— La version de Python utilisée pour corriger est la même qu'à l'IUT en salle du rdc : **python3.6** ! Code qui n'est pas dans la bonne version de Python → pénalité.

— Votre code doit être rendu au **format .txt** : pendant tout le projet, vous travaillerez avec un script `.py` mais au moment du rendu vous aurez juste à changer l'extension. Cela permettra d'utiliser le détecteur de plagiat Compilatio de l'Université.

— Vous avez **3x4 heure**s de projet en autonomie, cela veut dire que vous n'êtes pas encadrés par un enseignant. Si vous bloquez vraiment sur certains points ou que le cahier des charges n'est pas clair, contactez l'enseignant par mail.

1.2 Objectifs

Le but du projet est de travailler avec l'*API REST* **PokéAPI** en Python afin de produire diverses statistiques plus ou moins utiles. Enfin, on utilisera ces statistiques pour produire des rapports au format *Markdown*, puis en *HTML*. On veut réaliser :

1. Le projet **pokestats_Pokemon** : Avoir une page Web affichant les statistiques d'un Pokémon. Celle-ci contiendra son identifiant, son nom, une image du Pokémon (*sprite*), son poids, sa taille, et la liste des attaques qu'il peut apprendre. L'objectif final est juste d'avoir une méthode qui demande un identifiant de Pokémon et génère le fichier HTML.
2. Le projet **pokestats_heightHP** : Avoir une page Web affichant des statistiques spécifiques sur plusieurs Pokémons. On veut répondre à une question bête : les Pokémons les plus grands sont-ils ceux qui ont le plus de point de vie de base ? L'objectif final est d'avoir une méthode qui produit une page HTML avec une liste de Pokémons triés par ordre de taille, avec pour chacun leurs points de vie de base.

2 Fonctionnement général

→ La méthode `pokestats_Pokemon(id)` :

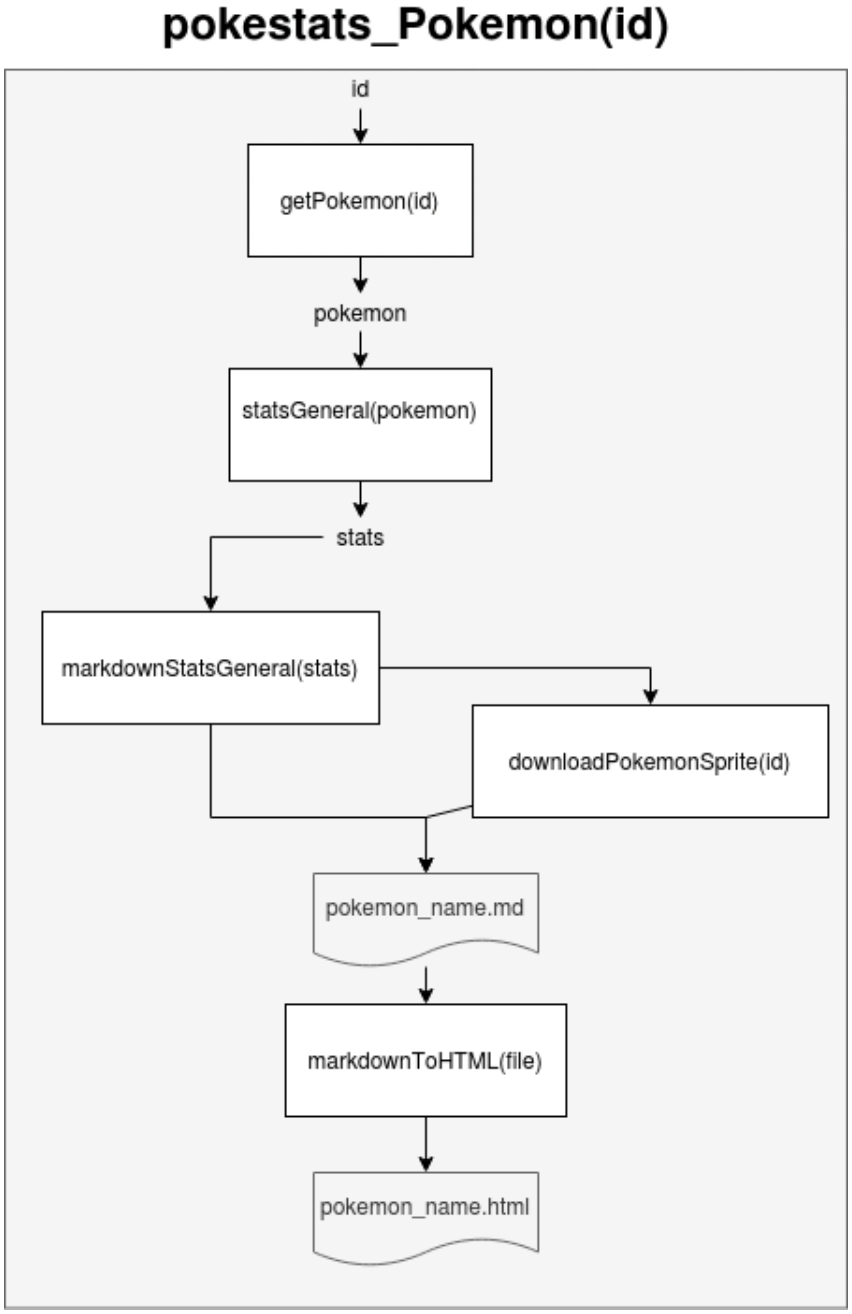
— prends en entrée un numéro de Pokémon

— va chercher un pokémon sur l'API avec `getPokemon(id)`

— récupère des stats sur ce Pokémon avec `statsGeneral(pokemon)`

— convertit ces stats dans un fichier `nom_pokemon.md` avec `markdownStatsGeneral(stats)` et télécharge aussi son image (*sprite*) avec `downloadPokemonSprite(id)`

— convertit enfin le fichier Mardown en HTML avec `markdownToHTML(file)`



→ La méthode `pokestats_heightHP(first, last)` :

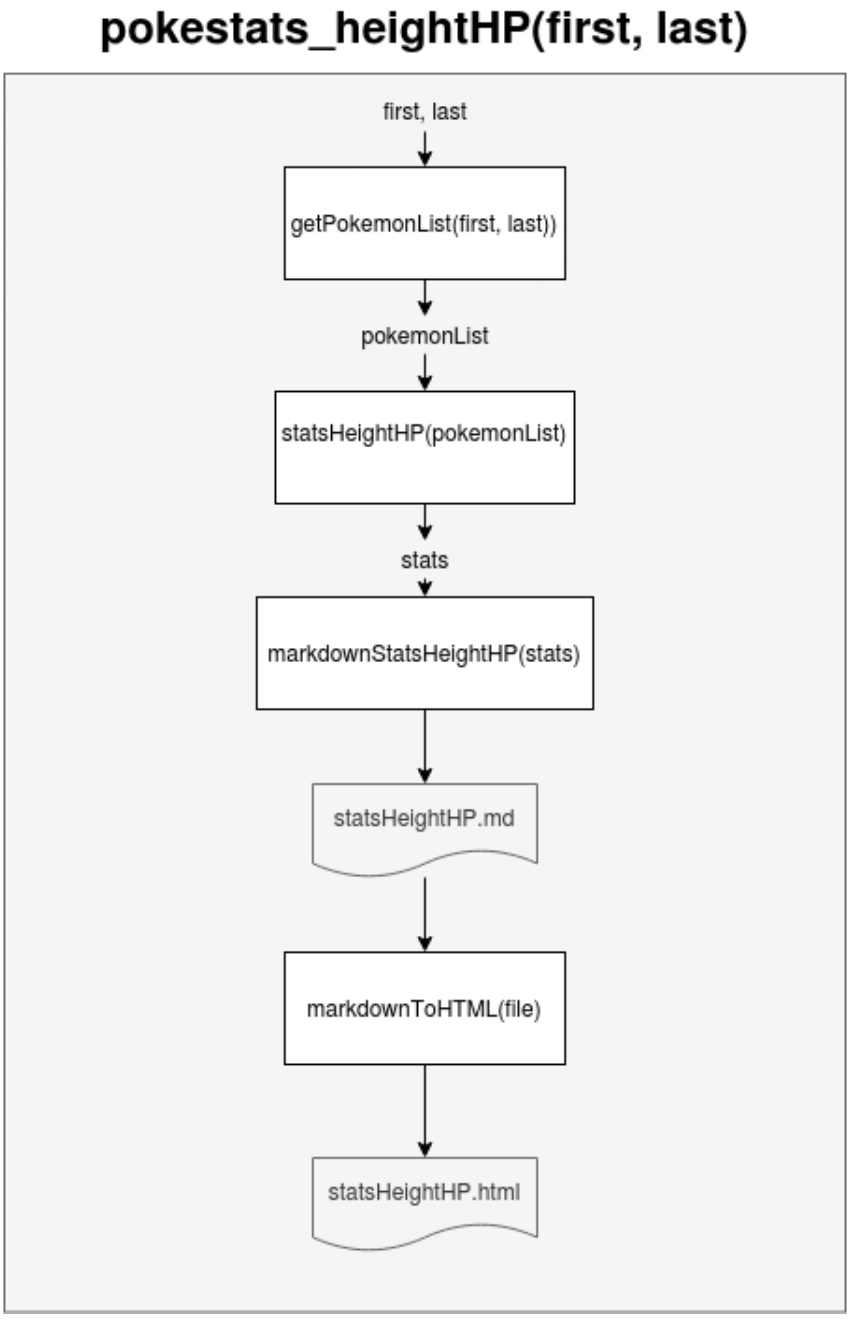
— prends en entrée deux numéros de Pokémon

— va chercher une liste pokémon sur l'API du premier au dernier passé avec `getPokemonList(first,`

— récupère des stats de taille et de points de vie sur ces Pokémons `statsHeightHP(pokemonList)`

— convertit ces stats dans un fichier `statsHeightHP.md` avec `markdownStatsHeightHP(stats)`

— convertit enfin le fichier Mardown en HTML avec `markdownToHTML(file)`



3 Méthodes à développer

Le programme principal fait appel aux deux méthodes présentées précédemment qui sont la logique générale du programme. Dans cette catégorie, vous trouverez le détail de chaque sous-méthode à réaliser : ce qu'elles attendent en entrée, leur fonctionnement, et des exemples d'exécution.

3.1 Collecte de données

- `getPokemon(id)` :
  - prends en entrée un identifiant de Pokémon (entier)
  - contacte l'API via le wrapper pokepy pour récupérer le Pokémon de l'identifiant donné dans une variable
  - renvoie la variable comme résultat de la méthode

```
>>> psy = getPokemon(54)
>>> print(psy)
<Pokemon - Psyduck>
```

- `getPokemonList(firstID, lastID)` :
  - prends en entrée deux identifiants de Pokémon (entiers)
  - pour chaque Pokémon de firstID à lastID, contacte l'API via le wrapper pokepy pour récupérer chaque Pokémon dans une variable
  - stocke chaque Pokémon dans une liste
  - renvoie la liste comme résultat de la méthode

```
>>> maliste = getPokemonList(7,18)
>>> print(maliste)
[<Pokemon - Squirtle>, <Pokemon - Wartortle>, <Pokemon - Blastoise>, <
    Pokemon - Caterpie>, <Pokemon - Metapod>, <Pokemon - Butterfree>, <
    Pokemon - Weedle>, <Pokemon - Kakuna>, <Pokemon - Beedrill>, <
    Pokemon - Pidgey>, <Pokemon - Pidgeotto>]
```

- `downloadPokemonSprite(id)` :
  - prends en entrée un identifiant de Pokémon (entier)
  - va récupérer via l'API l'URL du *sprite* (image) du Pokémon dans son format par défaut de face (*sprite front_default*)
  - télécharge l'image via l'URL et sauvegarde l'image
  - le chemin de sauvegarde de l'image est : `./sprites/pokemon_name.png` (on considère que le répertoire local `sprites` existe déjà)
  - ne renvoie pas de résultat

```
>>> downloadPokemonSprite(340)
# ne renvoie rien, après exécution on va trouver dans le rep local
    sprites une image du pokémon 340 nommée whiscash.png
```

3.2 Réalisation de statistiques

- `statsGeneral(pokemon)` :
  - prends en entrée un objet Pokémon (obtenu auparavant par `getPokemon()`)
  - crée une liste qui contient des stats sur le Pokémon passé en paramètre : son identifiant, son nom, sa hauteur, son poids, et une liste de ses attaques possibles
  - renvoie la liste des stats comme résultat

```
>>> pikachu = getPokemon(25)
>>> statsPika = statsGeneral(pikachu)
>>> print(statsPika)
[25, 'pikachu', 4, 60, ['mega-punch', ... , 'confide', 'nuzzle']]
# je n'ai pas mis toutes les attaques pour faciliter la relecture
```

- `statsHeightHP(pokemonList)` :
  - prends en entrée une liste de Pokémon (obtenue auparavant par `getPokemonList()`)
  - pour chaque élément dans pokemonList, ajoute dans une liste globale son nom, sa hauteur et ses points de vie
  - trie la liste de stats globale du plus petit pokémon au plus grand (en taille)
  - retourne la liste comme résultat

```
>>> listePokemons = getPokemonList(20,25)
>>> statsObtenues = statsHeightHP(listePokemons)
>>> print(statsObtenues)
[['spearow', 3, 40], ['raticate', 7, 55], ['fearow', 12, 65], ['ekans'
    , 20, 35], ['arbok', 35, 60]]
```

3.3 Mise en forme des stats avec Markdown et HTML

- `markdownStatsGeneral(stats)` :
  - prends en entrée une liste de statistiques sur un Pokémon (obtenue auparavant par `statsGeneral()`)
  - va télécharger le sprite du Pokémon avec `downloadPokemonSprite()`
  - prépare une variable contenant des données Markdown sur le Pokémon :
    - son nom est dans un titre principal
    - sa taile et son poids sont dans deux sous-titres respectifs
    - son sprite est inclut comme une image depuis le répertoire `sprites`
    - enfin, la liste de ses attaques possibles est donnée sous forme de liste non-ordonnée
  - écrit dans le répertoire courant dans un fichier `pokemonName_stats.md` les données au format Markdown
  - ne renvoie pas de résultat particulier

```
>>> pikachu = getPokemon(25)
>>> statsPika = statsGeneral(pikachu)
>>> markdownStatsGeneral(statsPika)
# ne renvoie rien, mais on peut trouver dans le rép local un fichier
    pikachu_stats.md contenant les données demandées au format Markdown
```

- `markdownStatsHeightHP(stats)` :
  - prends en en entrée une liste de statistiques sur des Pokémons (obtenue auparavant par `statsHeightHP()`
  - prépare une variable contenant les données au format Markdown :
    - un titre général
    - un tableau avec une en-tête "nom, taille, HP"
    - pour chaque élément dans la liste `stats` (donc chaque Pokémon) remplir le tableau avec le nom du Pokémon courant, sa taille et ses points de vie de base
  - écrit dans le répertoire courant un fichier `statsHeightHP.md` les données au format Markdown
  - ne renvoie pas de résultat particulier

```
>>> listePokemons = getPokemonList(20,25)
>>> statsObtenues = statsHeightHP(listePokemons)
>>> markdownStatsHeightHP(statsObtenues)
# ne renvoie rien, mais on peut trouver dans le rép local un fichier
    statsHeightHP.md contenant les données demandées au format Markdown
```

- `markdownToHTML(file)` :
  - prends un nom de fichier en entrée (le fichier Markdown à convertir)
  - transforme le contenu du Markdown en HTML, et l'écrit dans un fichier .html du même nom !

4 Annexes

4.1 Barème indicatif

— Méthode générale `pokestats_Pokemon(id)` et toutes ses sous-méthodes : sur **10 points**

— Méthode générale `pokestats_heightHP(first, last)` et toutes ses sous-méthodes : sur **8 points**

— Ingéniosité, qualité du code. Commentaires et documentation éventuelle : sur **2 points**

4.2 Exemples de résultats HTML

```
1 # exemple d'exécution
2 >>> pokestats_Pokemon(151)
3 # génère le fichier mew_stats.html avec le contenu suivant (je n'ai mis que
      le début des attaques) :
```

Nom du pokémon : mew

Taille : 4

Poids : 40

Liste des attaques :

- pound
- mega-punch
- pay-day
- fire-punch
- ice-punch
- thunder-punch
- razor-wind
- swords-dance
- cut
- whirlwind

```
1 # exemple d'exécution
2 >>> pokestats_heightHP(1,10)
3 # génère le fichier statsHeightHP.html avec le contenu suivant:
```

Les pokémons grands sont-ils les plus tanky ?

Nom	Taille	HP
squirtle	5	44
charmander	6	39
bulbasaur	7	45
ivysaur	10	60
wartortle	10	59
charmeleon	11	58
blastoise	16	79
charizard	17	78
venusaur	20	80