Formation Python, administration système
Correction des travaux pratiques

Matthieu Falce

Octobre 2025

1 Manipulation de la syntaxe python

1.1 *Fizz Buzz*

Voir le code de correction situé : `corrections/syntaxe/fizzBuzz.py`

1.2 Plus ou moins

Voir le code de correction situé : `corrections/syntaxe/plusOuMoins.py`

1.3 Slices

Voir le code de correction situé : `corrections/syntaxe/slices.py`

1.4 Magic 8 ball

Sous *Windows* il peut y avoir des soucis à l'ouverture du fichier à cause de l'encodage de caractère de cet *OS*. Il faut indiquer un encodage lors de l'ouverture du fichier dans ce cas.

Voir le code de correction situé : `corrections/syntaxe/magic8ball.py`

1.5 Comparaison de textes

Réponses :

1. on utilise la bibliothèque tierce `requests` (plus simple) ou ce qui existe dans la bibliothèque standard (meilleure portabilité)
2. on transforme la liste de mots en `set` (complexite $\mathcal{O}(n)$)
3. on transforme la liste de mots avec un `collections.Counter` (complexite $\mathcal{O}(n)$ je pense). Si l'on regroupe les textes, il faut *merger* les deux compteurs (ce sont des dictionnaires)
4. on utilise les opérations intersection des ensembles
5. on utilise un compteur en lui indiquant la limite à afficher

Voir le code de correction situé : `corrections/syntaxe/comparaisonTextesInternet.py`

1.6 C'est looooong

Réponses :

1. on utilise `time.sleep(3)` pour mettre en pause l'execution de 3s

2. on utilise `time.time()` pour marquer le début et la fin de la zone à mesurer, on soustrait les valeurs et voilà
3. avec le décorateur on peut appeler plusieurs fois la fonction pour calculer moyenne et écart-type de la durée
4. il y a `timeit`

Voir le code de correction situé : `corrections/syntaxe/mesureTemps.py`

1.7 Analyse de complexité

Réponses :

1. on va générer des conteneurs de taille différentes, ensuite nous allons regarder le temps que met l'opération pour les différentes tailles, pour chaque type de conteneurs
2. nous allons utiliser `time.time` pour mesurer le temps d'éxecution de la fonction d'intérêt. En réalité, cela est plus complexe à effectuer correctement et devrait effectuer une analyse statistique sur de nombreuses répétitions pour lisser l'utilisation de l'ordinateur
3. on est cohérent avec les complexités notées par la doc (les structures hashées *dict* et *set* ont des temps d'accès linéaires)
4. on utilise la bibliothèque tierce `matplotlib` sur nos différentes séries

Voir le code de correction situé : `corrections/syntaxe/complexite/analyseComplexite.py`

2 Programmation orientée objet

2.1 Formation

Voir le code de correction situé : `corrections/poo/formation.py`

2.2 Convertisseur de température

Réponses :

1. c'est une mauvaise idée de dupliquer les informations, quelle est la source de vérité ? En plus, en python, le manque d'encapsulation permet de modifier les deux attributs indépendamment, ce qui peut complètement gêner le code
2. `property` c'est la vie 😉. On peut simuler des getters et setters en les manipulant comme des attributs

Voir le code de correction situé : `corrections/poo/convertisseur.py`

3 Modules

3.1 Bases de données

Réponses :

1. on peut en stocker autant que nécessaire pour la fonctionnalité métier. Dans ce cas, stocker, le nom, matière et note semble etre un minimum
2. on utilise du SQL pour effectuer les recherches (filtres, accès), on peut également utiliser un `ORM` *object relationship manager* pour ne pas avoir à écrire de code

Voir le code de correction situé : `corrections/modules/bdd/sqlite.py`

Voir le code de correction situé : `corrections/modules/bdd/sqlalchemyImplementation.py` (pour l'utilisation d'un `ORM`)

3.2 Expressions régulières

Tous, trouvez les tous

Réponses :

1. on parse le fichier en utilisant une expression régulière
2. une fois parsé, on met les informations dans un `collection.Counter`

Voir le code de correction situé : `corrections/modules/regexp/versions.py`

Cherchez / trouvez

Réponses :

1. on met une option sur la lettre "h"
2. on met une contrainte sur le début et la fin du mot (par exemple, il faut que ce soit un espace ou le début / fin de ligne)
3. non, quand on commence à construire des expressions trop complexes, il peut être intéressant de développer sa propre solution en python (le langage permettant de facilement travailler les chaînes de caractères)

Voir le code de correction situé : `corrections/modules/regexp/chat.py`

Identifiants

Réponses :

1. toutes ces chaînes sont structurées de la même façon
2. on peut donc utiliser une expression régulière pour capturer les différents groupes

Voir le code de correction situé : `corrections/modules/regexp/extractUUID.py`

C'est valide

Réponses :

1. une adresse email doit ressembler à une forme canonique. Attention, cela n'est pas la bonne façon de faire, pour être sûr que le mail existe, il faut l'envoyer, les serveurs mails se chargeront de chercher à le distribuer.
2. on peut créer une expression régulière permettant de vérifier ses informations
3. on ouvre le JSON et on extrait des groupes de capture

Voir le code de correction situé : `corrections/modules/regexp/validationEmails.py`

4 Administration système

Cette section comporte une sorte de projet pour faire découvrir la puissance de python dans le cadre de l'administration système (manipulation de textes, de fichiers, de scripts...).

4.1 Manipulation de chaines de charactères

Vous pouvez trouver la correction ici :

Voir le code de correction situé : `corrections/administrationSysteme/manipulationTexte/meilisearchExtraction.py`

4.2 Manipulation de fichiers

Vous pouvez trouver la correction ici :

Voir le code de correction situé : `corrections/administrationSysteme/manipulationFichiers/main.py`

4.3 Manipulation d'adresse IP

Vous pouvez trouver la correction ici :

Voir le code de correction situé : `corrections/administrationSysteme/IP/ipPurPython.py`

Réponses :

1. nous devons surcharger les méthodes magiques `__contains__` et `__iter__`
2. utiliser la bibliothèque `ipadress` ne change pas l'API de notre classe. C'est tout l'intérêt de l'orienté objet (séparer l'appel et l'implémentation). On peut cependant se demander *l'intérêt de notre classe* vue l'existence de `ipadress`.

La version utilisant la bibliothèque se trouve ici :

Voir le code de correction situé : `corrections/administrationSysteme/IP/ipLibrary.py`

4.4 Manipulation de logs

Vous pouvez trouver la correction ici :

Voir le code de correction situé : `corrections/administrationSysteme/manipulationLogs/apache.py`

4.5 Extraction de données (`scrapping`)

Pour `scrapy` :

Voir le code de correction situé : `corrections/administrationSysteme/scraping/scrapingproject/scrapingproject/spiders/ourfirstbot.py`

Réponses :

1. pour rechercher les éléments d'intérêt il y a trois solutions majoritaire. Les navigateurs et leur options de debug permettent d'y accéder facilement
   - nous parcourons les enfants et parents HTML à la main (assez fragile s'il y a un changement de structure de la page)
   - nous utilisons le XPATH qui permet de décrire la relation parent / enfant du HTML de façon normalisée (mêmes problèmes qu'au dessus)
   - nous récupérons les éléments selon leur classe / identifiant CSS (plus solide car utilisé pour le style de la page)
2. la réponse dépend du besoin
   - globalement, `requests` est une bibliothèque bas niveau qui permet d'interagir avec un serveur HTTP, le parsing et l'extraction d'information doivent être écrits par les développeurs spécialement pour chaque page
   - alors que `scrapy` est un framework de scraping qui intègre toutes ces mécaniques nativement
   - à mon sens, `scrapy` peut être plus difficile à mettre en place au début (il faut que le développeur s'adapte au framework) mais permet d'être plus rapide une fois cette phase passée