Formation Python, perfectionnement
Correction des travaux pratiques

Matthieu Falce

Janvier 2023

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 Fonctions avancées

Cette section comporte des exercices qui mettent en pratique les manipulations avancées de fonctions

2.1 Miam

Réponses :

1. le plus proche de la fonction est appelé en premier
2. oui on peut obtenir le même comportement, il faut indiquer le paramétrer sur le texte et la position avant ou apres

Voir le code de correction situé : `corrections/fonctionsAvancees/miam.py`

2.2 Déprécié

Réponses :

1. cela dépend de ce que l'on veut faire :
   — avant le wrapper, on les stocke toutes
   — dans le wrapper, on ne stocke que les fonctions appellées

Voir le code de correction situé : `corrections/fonctionsAvancees/deprecie.py`

2.3 *Fizz Buzz* is back

Réponses :

1. globalement non, les solutions sont compliquées à comprendre et n'apportent pas de grands avantages

Voir le code de correction situé : `corrections/fonctionsAvancees/lazyFizzBuzz.py`

2.4 Pipeline grep

Réponses :

1. le flux étant par définition continue et infini, nous devons utiliser un pipeline qui tire les données, donc un pipeline de consommateurs

Voir le code de correction situé : `corrections/fonctionsAvancees/pipelineConsommateur.py`

2.5 Pipeline filtres

Réponses :

1. on utilise le `design pattern Chain Of Responsability` qui permet d'ajouter ou d'enlever les filtres facilement

Voir le code de correction situé : `corrections/fonctionsAvancees/pipelineGenerateur.py`

2.6 ItertoolsBis

Voir le code de correction situé : `corrections/fonctionsAvancees/itertoolsBis.py`

2.7 Pareisseux

Réponses :

1. pour les manipulations de données dans l'ordre, `itertools` est pertinent. Cependant, pour manipuler à l'envers il ne peut rien faire.

Voir le code de correction situé : `corrections/fonctionsAvancees/sliceParaisse.py`

2.8 C'est loooong

Voir le code de correction situé : `corrections/fonctionsAvancees/looong.py`

3 Programmation orientée objet

3.1 Formation

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

3.2 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`

4 POO avancée

Cette section comporte des exercices qui mettent en pratique les manipulations avancées d'objets

4.1 Pathlib

Voir le code de correction situé : `corrections/pooAvancee/path.py`

4.2 Properties

Voir le code de correction situé : `corrections/pooAvancee/property.py`

4.3 La classe, la méta-classe

Réponses :

1. le ___new___ de la métaclasse n'est pas appelé, c'est celui de la classe qui l'est
2. le ___call___ de la classe n'est pas appelé, c'est celui de la métaclasse qui l'est

Voir le code de correction situé : `corrections/pooAvancee/metaclasses.py`

4.4 Singleton

Réponses :

1. cela dépend de la façon dont on s'y prend. On peut modifier le ___new___ d'une classe ou le ___call___ d'une méta classe
2. on utilise l'attribut `metaclass` de `object`
3. c'est une question de point de vue...

Voir le code de correction situé : `corrections/pooAvancee/singletonMetaclasse.py`

5 Modules

5.1 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`)

5.2 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`

6 Parallélisme et performances

On va regarder ici différentes méthodes permettant d'améliorer les performances d'une application.

6.1 Mise en cache

Réponses :

1. on ne peut pas gérer la taille du cache, ...
2. la fonction de la bibliothèque standard est mieux

Voir le code de correction situé : `corrections/parallelisme/fibonacci.py`

6.2 Travailleurs !

Réponses :

1. les opérations sont IO bounds à priori, les performances devraient être équivalentes
2. on utilise des `pools` et on met des limites dans le code qui appelle. Je ne suis pas sûr qu'il existe de meilleures méthodes...

Voir le code de correction situé : `corrections/parallelisme/workers.py`

7 Intégration Python / C

7.1 Intégration code natif

Réponses :

1. il faut mesurer pour le savoir. L'overhead n'est pas anodin il me semble.
2. cython permet de manipuler les `numpy.array` facilement

Voir le code de correction situé : `corrections/integrationC/ctypes/main.py`

Voir le code de correction situé : `corrections/integrationC/cython/main.py`
Les codes a utiliser pour compiler se trouvent :

— `corrections/integrationC/ctypes/readme.md`

— `corrections/integrationC/cython/readme.md`

7.2 On embarque

Voir le code de correction situé : `corrections/integrationC/embedPython/readme.md`

8 Utilisation en *datascience*

8.1 Analyse Covid

Voir le code de correction situé : `corrections/scipy/Covid.ipynb`