Formation Python, programmation Objet
Correction des travaux pratiques

Matthieu Falce

Novembre 2022

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 Fils rouges

Cette section comporte des exercices qui sont transverses à plusieurs parties ou qui nécessitent d'avoir une vision plus globale de l'architecture d'un projet python.

4.1 On reprend tout

Voir le code de correction situé : `corrections/filRouge/reprendTout/main.py`

4.2 Un géant aux pieds d'argiles

Il y a plusieurs étapes à la résolution de ce problème. Il faut y aller par étapes. L'important est de garder à l'esprit que chaque étape ne doit pas fragiliser les autres (d'où l'importance des tests unitaires) et surtout il faut décider quand s'arrêter.

Voir le code de correction situé : `corrections/filRouge/refactoring/fonctionnel.py`

Voir le code de correction situé : `corrections/filRouge/refactoring/mieux.py`

Voir le code de correction situé : `corrections/filRouge/refactoring/final/readme.md`

5 Interfaces graphiques avec Tkinter

5.1 Des boutons partouts

Réponses :

1. deux solutions se présentent
   - la plus économe (celle que l'on va mettre en oeuvre) : utiliser `eval` (cela peut poser des problèmes de sécurité)
   - la plus évoluée : implémenter son parseur d'opérations mathématiques
2. on va utiliser un `gridlayout` , une calculette pouvant se résumer à une grille de boutons
3. on fait en sorte que chaque bouton rajoute des caractères à une chaîne que l'on évaluera

Voir le code de correction situé : `corrections/gui/calculatrice.py`

5.2 C'est pas clair tout ça

Réponses :

1. je laisse libre cours à votre imagination. Un slider ou un bouton affichant un numéro aléatoire à chaque fois semble particulièrement diabolique.
2. la meilleure interface (pour un numéro français) serait de regrouper les chiffres deux par deux dans un `text_input`

Voir le code de correction situé : `corrections/gui/longTasks.py`

5.3 Pitichat

Réponses :

1. le système est asynchrone, il faut que le tâches soient très rapides pour ne pas géner la boucle d'exécution principale. Le téléchargement d'une ressource depuis internet est trop longue, nous réalisons alors le blocage de la boucle
2. on peut mettre un `time.sleep(100)` plutôt que de télécharger pour reproduire simplement le problème. Cela confirme l'hypothèse de la tâche trop longue
3. il faut déléguer les tâches longues dans une `thread`

Voir le code de correction situé : `corrections/gui/longTasks.py`

5.4 Artistes

Réponses :

1. on utilise un `canvas` tkinter
2. on va modifier la variable utilisée pour déterminer la couleurs du trait
3. on choisi le chemin avec `filedialog.asksaveasfilename` et on sauvegarde avec `Canvas.postscript`

Voir le code de correction situé : `corrections/gui/calculatrice.py`

5.5 Je table sur ça

Réponses :

1. la source de vérité est les données ainsi que les filtres appliqués. Dès que l'un ou l'autre change, on réaffiche les données
2. cela peut provoquer des artefacts
   — on peut éviter de rafraîchir trop souvent les infos (mettre un délais entre deux mise à jours de l'affichage)
   — on peut mettre en cache les informations

Voir le code de correction situé : `corrections/gui/tableur.py`

6 Intégration Python / C

6.1 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`

6.2 On embarque

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