Projet

…

D. Projet [Partie 2]

Clean code et linter

Les linters en python, sont des programmes qui identifient les sources potentielles d'erreurs dans un code. Ces erreurs peuvent être des erreurs :

- de style ;
- de logique ;
- de syntaxe ;
- d'indentation ou des fautes de frappe…

Un linter en est un programme qui permet de vérifier le style des fichiers de code, et notamment qu'ils respectent les conventions et les règles énoncées par PEP 8. En d'autres termes, le linter vous aidera grandement à déceler à l'avance des erreurs de « codage » mais également de clean code. Les Linters sont divers et variés mais nous utiliserons « Pylint ». Pylint est intégré à beaucoup d'éditeurs. Par exemple, il est disponible comme add-on sur VS Code. Vous pouvez également l'installer avec « pip install pylint ».
Utilisez Pylint et respectez les remarques qu'il vous fait. Nous tiendrons compte de son avis le jour de l'examen 😉

Isoler une journée

Implémentez la fonction logs_by_day qui permet de récupérer uniquement les logs d'une journée.

```
def logs_by_day(logs, day):
  """ Pre :
       - logs est une liste où chaque élément est une ligne de log bien formée
       - day (str) est une date au format « Moi JJ »
      Post :
       - retourne une liste de logs qui concernent uniquement le jour
correspondant à day.
  """
```

Tester

Pensez à télécharger le nouveau fichier de test (test_main) sur Moodle. Il contient les tests de la partie 2 en plus des premiers !

Comparer des dates

Il serait intéressant de pouvoir extraire d'une grande liste de logs uniquement ceux entre deux moments précis. Pour ce faire, il va falloir comparer des dates et heure entre elles.
Le format actuel des heures dans le fichier de log nous permet de facilement les comparer entre elles. En effet, 4h18 et 23 secondes est plus tôt que 13h12 et 42 secondes et, en python, « 04:18:23 » est bien plus petit que « 13:12:42 ».
En revanche, concernant les dates, le format actuel ne permet pas de les comparer. En effet, le 1er décembre est après le 18 octobre alors que, en python, « Dec 01 » est plus petit que « Oct 18 ». Il faudra donc convertir les dates en un format qui permet la comparaison. Différents choix peuvent être fait et beaucoup seraient tout à fait valide mais nous devons nous mettre d'accord sur un. Nous choisirons donc un format semblable à celui utilisé pour l'heure afin d'uniformiser un maximum. Nous transformerons donc nos dates en suivant le format « MM:JJ » ou MM est le mois en 2 chiffres et JJ le jour en 2 chiffres. « Dec 01 » deviendrait donc « 12:01 » et « Oct 18 » deviendrait « 10:18 ». Dès lors, « 12:01 » sera bien plus grand que « 10:18 ».

Implémentez la fonction suivante :

```
def formated_date(date):
    """
    Pre : date (str) est la date au format "Mois JJ HH:MM:SS" où :
                - Mois : est les 3 première lettres du mois en anglais
                    (commencant par une majuscule)
                - JJ : est le numéro du jour (sur 2 chiffres)
                - HH : est l'heure (sur 2 chiffres)
                - MM : sont les minutes (sur 2 chiffres)
                - SS : sont les secondes (sur 2 chiffres)
    Post : Retourne la date sous le format "XX:JJ HH:MM:SS"
                où XX est le nombre du mois (sur 2 chiffres)
    """
```

Testez-la et corrigez-les si besoin.

Isoler une plage horaire

Implémentez la fonction log_between qui permet de récupérer uniquement les logs entre deux dates.

```
def logs_between(logs, date_min="00:00 00:00:00", date_max="99:99 00:00:00"):
  """ Pre :
       - logs est une liste où chaque élément est une ligne de log bien formée
       - date_min et _max sont des str représentant une date (et heure) au
format "XX:JJ HH:MM:SS". Par défaut, ils ont de valeurs absurdes permettant de
prendre tout.
       - date_min est plus petit que date_max
      Post :
       - retourne une liste de logs qui concernent des dates entre date_min et
date_max (inclus).
  """
```