Projet

…

E. Projet [Partie 3]

Linter

Lors de la partie 2, nous avons abordé les linters avec Pylint. Celui-ci peut ressortir un grand nombre de problèmes différents et tous n'ont pas la même importance.

Pour rappel, votre note finale dépendra des tests unitaires mais aussi du linter. Cependant, toutes les remarques du linter n'affecteront pas votre note de la même façon. Voici une idée de l'importance de chaque type de remarque :

- Remarques mettant automatiquement en échec :
  - Noms des variables/fonctions
  - Docstrings de fonctions
  - Lignes trop longues
- Remarques faisant perdre quelques points :
  - Docstring du module
  - Gestion des espaces
- Remarques ignorées :
  - Exceptions trop large (car c'est de la matière non abordée)
  - Redéfinition d'une variable du scope englobant
  - Utilisation de « open » sans spécifier l'encodage

Faire ressortir les erreurs

Le but des scripts autour d'un fichier de log est de pouvoir mettre en avant des soucis. Faisons donc une fonction permettent d'isoler les logs d'erreur ou, de manière plus générale, qui contiennent un(des) mot(s) particulier(s) dans leur message. Écrivez la fonction `logs_with_tag` respectant les spécifications suivantes.

```
def logs_with_tag(logs, tag="error"):
    """ Pre :
       - logs est une liste où chaque élément est une ligne de log bien formée
       - tag est une chaine de caractères à trouver dans le message.
            Par défaut : "error"
    Post :
       - retourne une liste de logs qui concernent uniquement des logs
            contenant le tag (minuscule ou majuscule) dans le message
    """
```

Remarque

On veut que le tag soit en minuscule ou en majuscule dans le message. Pour se faire, la fonction `lower()` peut vous aider. Elle permet de mettre une chaine de caractères en minuscule. Par exemple, `"Coucou Bob".lower()` retournera « coucou bob ».

Des informations par rapport à un programme

Faisons en sorte également d'isoler les logs concernant un programme en particulier grâce à la fonction `logs_from_program`.

```
def logs_from_program(logs, program):
    """ Pre :
       - logs est une liste où chaque élément est une ligne de log bien formée
       - program (str) est le programme à trouver
    Post :
       - retourne une liste de logs qui concernent uniquement les programmes
correspondant à "program"
    """
```

Prévoyons également la fonction `list_process_for_program` qui liste les ids des différents processus gérés par le programme passé en paramètre. Attention que cette liste ne doit contenir aucun doublon.

```
def list_process_for_program(logs, program):
    """ Pre :
       - logs est une liste où chaque élément est une ligne de log bien formée
       - program (str) est le programme à trouver
    Post :
       - retourne une liste des process_id gérés par le programme.
            La liste ne contient aucun doublon.
    """
```

> Remarques :
>
> N'oubliez pas que certains programmes n'ont pas de processus (comme kernel par exemple). Dans ce cas, sa liste des processus sera vide.
>
> Un programme a soit toujours des processus soit aucun… mais aucun programme n'a parfois des processus et parfois pas.

Les programmes suspects

Implémentez la fonction `suspects` qui retourne une liste des programmes ayant généré des logs d'erreur.

```
def suspects(logs, limit):
    """ Pre :
       - logs est une liste où chaque élément est une ligne de log bien formée
       - limit est le nombre limite d'erreurs tolérées pour un programme
    Post :
       - retourne une liste des programmes (sans doublons) qui ont généré
       plus que le nombre limite de log signalant des erreurs (error).
    """
```