Examen de TP, fin du premier semestre

Récupérez votre dossier d'étude dans GitHubDesktop et ouvrez dans Pyzo le fichier `.py` commençant par `TP_note_sem1`.

N'oubliez pas que l'exécution doit se faire via `Ctrl-SHIFT-E` pour que tout marche bien.

Partie I

Travail sur les listes

Dans toute cette section, on vous fournit une fonction `voulu(elem)` qui détermine si l'élement `elem` est voulu ou non. Elle renvoie `True` si c'est le cas et `False` sinon.

On vous demande d'écrire quelques fonctions qui agissent sur une liste :

1. `au_moins_un_voulu(liste)` qui renvoie `True` s'il y a au moins un élément voulu dans la liste et `False` sinon.
2. `compte_nombre_voulus(liste)` qui compte le nombre d'éléments voulus dans la liste.
3. `rassemble_elements_voulus(liste)` qui renvoie une liste qui contient tous les éléments voulus d'une liste donnée.

HINT : il y a moyen d'écrire simplement la première fonction en utilisant la deuxième et d'écrire simplement la deuxième en utilisant la troisième... À vous de voir ce que vous préférez.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie II

Le juste prix

Nous allons jouer à un « remake » du Juste Prix. Dans notre jeu, un animateur veut vous faire deviner le prix (en euros, sans centimes) d'un quelconque bibelot. Pour vous aider, le public (qui a été mis dans le coup) peut vous souffler `'plus'` si votre estimation est trop basse, `'moins'` si votre estimation est trop haute et vous dira `'bravo'` si vous avez trouvé la bonne valeur. Écrire une fonction `juste_prix` qui prend en argument le prix maximum `prix_max` envisageable dans le jeu (il ne peut pas y avoir de valeur négative), le nombre `N` d'essais que vous avez à votre disposition (attention à ne pas dépasser) et la fonction `public` qui vous permettra de demander l'avis du public. Votre fonction doit renvoyer le prix qui correspond aux attentes du public. Un squelette vous est proposé dans le fichier afin de s'assurer que vous ne demandiez pas l'avis du public plus de `N` fois.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie III

Parcours d'une arborescence

Les ordinateurs sont conçus en utilisant un système à arborescence de fichiers. On dispose en fait de deux types d'objets :

— les simples fichiers avec des extensions qui précisent leurs types comme `.pdf`, `.py`, `.mp4`, etc.

— les dossiers qui peuvent eux-mêmes contenir des fichiers ou d'autres dossiers (qui peuvent eux-mêmes contenir des fichiers ou d'autres dossiers (qui peuvent eux-mêmes [1]...))

On désire écrire une fonction `recherche_fichiers(chemin, extension)` qui renvoie la liste des chemin relatifs vers tous les fichiers du type `extension` qui existe dans le dossier `chemin` (la liste doit être vide si `chemin` est un simple fichier qui n'a pas la bonne extension et non un dossier).

Par exemple, si on a l'arborescence suivante

```
exemple
|-- dummy.txt
|-- latex
|   |-- DS
|   |   |-- DS01.pdf
|   |   `-- DS01.tex
|   |-- note.pdf
|   `-- note.tex
|-- pdf
|   |-- DS01.pdf
|   `-- note.pdf
`-- python
    |-- induction.py
    `-- meca.py
```

alors les appels à la fonction demandée doivent donner le résultat suivant (l'ordre de renvoie des fichiers dans la liste n'est pas imposé)

```
>>> recherche_fichiers('exemple', 'pdf')
['exemple/latex/DS/DS01.pdf', 'exemple/latex/note.pdf', 'exemple/pdf/DS01.pdf',
'exemple/pdf/note.pdf']
>>> recherche_fichiers('exemple/latex', 'pdf')
['exemple/latex/DS/DS01.pdf', 'exemple/latex/note.pdf']
>>> recherche_fichiers('exemple', 'py')
['exemple/python/induction.py', 'exemple/python/meca.py']
>>> recherche_fichiers('exemple', 'mp4')
[]
>>> recherche_fichiers('exemple/dummy.txt', 'txt')
['exemple/dummy.txt']
>>> recherche_fichiers('exemple/dummy.txt', 'py')
[]
```

Pour vous aider dans votre quête, on vous fournit un certain nombre de fonction qu'il faudra forcément utiliser (à l'exclusion de toute fonction directement fournie par le système dans un quelconque module que vous seriez tenté d'importer suite à une recherche sur google) :

1. Vous la voyez venir la récursivité ?

— `est_un_fichier(chemin)` : vérifie si `chemin` mène vers un objet de type fichier. Il renvoie `True` si `chemin` est bien un fichier et `False` s'il s'agit d'un dossier.

```
>>> est_un_fichier('exemple')
False
>>> est_un_fichier('exemple/dummy.txt')
True
```

— `liste_contenu(chemin)` : si `chemin` est bien un dossier, renvoie une liste des chemins vers tous les objets (fichiers ou dossiers) qu'il contient. Meurt dans d'atroces souffrances si vous lui donnez un simple fichier en argument.

```
>>> liste_contenu('exemple/latex')
['exemple/latex/DS', 'exemple/latex/note.pdf', 'exemple/latex/note.tex']
>>> liste_contenu('exemple/dummy.txt')
NoRepositoryError
```

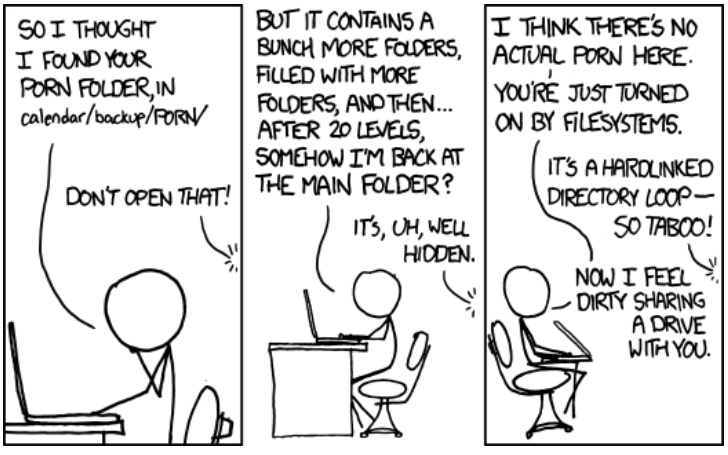
— `donne_extension(chemin)` : si `chemin` est bien un fichier, renvoie une chaîne de trois caractères ou moins qui correspond à l'extension (par exemple, `'txt'`, `'pdf'`, `'tex'` ou `'py'`, mais il peut y en avoir d'autres). Meurt dans d'atroces souffrances si vous lui donnez un dossier en argument.

```
>>> donne_extension('exemple/dummy.txt')
'txt'
>>> donne_extension('exemple')
NoFileError
```

Au vu de la structure, cela se prête bien à un traitement récursif (en réutilisant la fonction sur tous les dossiers que vous trouvez dans le dossier courant), mais ce n'est pas obligatoire. On peut très bien se prendre une liste annexe `a_visiter` dans laquelle on stocke au fur et à mesure les dossiers qu'on croise [2]. À vous de voir ce que vous préférez.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.



`Eww, gross, you modified link()? How could you enjoy abusing a filesystem like that?`

2. En itérant sous la forme `for dossier in a_visiter`, on peut remettre de nouveaux éléments dans `a_visiter` de manière à ce que la boucle se continue jusqu'à épuisement de la liste en question.

Partie IV

Montagnes russes

Vous venez d'être affecté au centre d'analyse et de supervision d'un nouveau parc d'attraction. Votre mission est d'estimer chaque jour quelle vont être les recettes de la journée pour chaque manège. Vous commencez par vous intéresser aux montagnes russes.

Vous remarquez que les montagnes russes plaisent tellement aux gens que dès qu'ils ont fini un tour de manège, ils ne peuvent s'empêcher de revenir pour un nouveau tour.

— Les personnes viennent faire la queue devant l'attraction.

— Elles peuvent soit être seules, soit être en groupe. Lorsque des groupes sont dans la queue, ils veulent forcément monter ensemble à bord, sans être séparés.

— Les personnes ne se doublent jamais dans la file d'attente.

— Dès qu'il n'y a plus assez de place dans l'attraction pour le prochain groupe dans la queue, le manège démarre. (Il n'est donc pas toujours plein).

— Dès que le tour de manège est terminé, les groupes qui en sortent retournent dans la file d'attente dans le même ordre.

Votre fonction `montagnes_russes(nb_places, nb_tours, groupes)` prend trois paramètre en argument :

— Le nombre `nb_places` de places qu'il y a dans un tour de manège.

— Le nombre `nb_tours` de tours de manège qu'il peut y avoir au total dans la journée.

— La liste `groupes` des groupes de personnes dans la file lors de l'ouverture de l'attraction (personne ne s'y rajoute par la suite et dès qu'ils sortent du manège, ils se remettent dans la file dans le même ordre). À noter qu'un même groupe ne peut pas être deux fois dans le même manège...

Votre fonction doit renvoyer un entier qui est le nombre total de personne qui auront pris le manège dans la journée.

```
>>> montagnes_russes(5, 3, [2, 3, 5, 3]) # Train plein à chaque fois ici...
15
>>> montagnes_russes(5, 5, [2, 3, 5, 3]) # ...mais ce n'est pas toujours le cas
23
>>> montagnes_russes(20, 3, [2, 3, 5, 3])# Tout le monde monte à tous les coups
39
>>> montagnes_russes(5, 4, [1, 5, 5, 1]) # Groupes déséquilibrés
13
>>> montagnes_russes(5, 4, [3, 8, 3, 1]) # Cas particulier où un groupe fait blocus
3
```

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.