Devoir surveillé d'informatique (1h30)

Exercice I

On veut construire une liste nommée `mult5` qui contient tous les multiples de 5 compris entre 30 et 100 inclus. On envisage trois méthodes :

Q 1 Méthode 1 : créer la liste `mult5` avec une boucle `for`.

▶ **Réponse :**

```
mult5 = []
for n in range(30,101):
    if n%5 == 0:
        mult5.append(n)
```

Q 2 Méthode 2 : créer la liste `mult5` avec une compréhension de liste.

▶ **Réponse :**

```
mult5 = [n for n in range(30,101) if n%5 == 0]
```

Q 3 Méthode 3 : créer la liste `mult5` en créant d'abord une liste d'entiers nommée `entiers`, puis en extrayant de celle-ci les nombres souhaités par *slicing*.

▶ **Réponse :**

```
entiers = list(range(30,101))
mult5 = entiers[::5]
```

Exercice II

Dans cet exercice on considère un jeu de labyrinthe, dont le plateau sera modélisé par un tableau à deux dimensions avec 10 colonnes et 5 lignes. Les cases du labyrinthe peuvent être de trois types :

- les cases vides, notées `'V'` ;
- les murs, notés `'M'` ;
- les cases spéciales, notées `'D'` pour la case de départ et `'A'` pour la case d'arrivée.

L'objectif de cet exercice est d'initialiser le plateau de jeu en plaçant les murs sur les bords du plateau, les cases spéciales en haut à gauche et en bas à droite, et les cases vides partout ailleurs comme indiqué sur la figure ci-dessous *(Les autres murs ne seront pas traités ici...)* :

M	M	M	M	M	M	M	M	M	M
M	V	V	V	V	V	V	V	V	M
M	V	V	V	V	V	V	V	V	M
M	V	V	V	V	V	V	V	V	M
M	M	M	M	M	M	M	M	M	M

On a crée un `tableau numpy` nommé `plateau` avec la commande suivante. Celle-ci crée un tableau vide de la taille demandée dont toutes les cases seront des chaînes de caractères :

```
plateau = np.empty((5, 10), dtype=str)
```

Q 4 Donner un code **utilisant des slices** permettant de remplir toutes les cases `M` du plateau de jeu précédent.

▶ Réponse :

```
plateau[0,:]='M'
plateau[-1,:]='M'
plateau[:,0]='M'
plateau[:,-1]='M'
```

Q 5 Saisir ensuite les `V` . Si on ne sait pas le faire avec des slices, envisager une solution avec `for`.

▶ Réponse :

```
plateau[1:-1,1:-1]='V'
```

Q 6 Enfin, donner un code pour remplacer les `M` des cases (1,0) et (3,9) respectivement par les cases spéciales `D` et `A` qui symbolisent le départ et l'arrivée.

▶ Réponse :

```
plateau[1,0]='D'
plateau[-2,-1]='A'
```

Exercice III

Dans cet exercice, on appelle **parenthèses** les couples de caractères ( ), { } et [ ].

Pour chaque couple de parenthèses, la première parenthèse est appelée la parenthèse ouvrante du couple et la seconde est appelée la parenthèse fermante du couple.

♪ Définition

On dit qu'une expression (chaîne de caractères) est bien parenthésée si :

- chaque parenthèse ouvrante correspond à une parenthèse fermante de même type ;
- les expressions comprises entre parenthèses sont des expressions bien parenthésées.

Exemple

- l'expression `'tab[2*(i + 4)] - tab[3]'` est bien parenthésée.
- l'expression `'tab[2*(i + 4] - tab[3)'` n'est pas bien parenthésée, car la première parenthèse fermante] devrait correspondre à la dernière parenthèse ouvrante (.

Q 7 Déterminer si l'expression `'[2*(i+1)-3) for i in range(3, 10)]'` est bien parenthésée. Justifier votre réponse.

▶ Réponse :

L'expression n'est pas bien parenthésée, car la deuxième parenthèse fermante) devrait correspondre à la première parenthèse ouvrante [: `'[2*(i+1)-3)'`.

Partie A : une condition nécessaire

On peut observer que, puisque les parenthèses vont par couple, une expression bien parenthésée contient autant de parenthèses ouvrantes que de parenthèses fermantes.

Q 8 Écrire une fonction `compte_ouvrante(txt : str) -> int` qui prend en paramètre une chaîne de caractères `txt` et qui renvoie le nombre de parenthèses ouvrantes qu'il contient.

▶ Réponse :

```
def compte_ouvrante(txt : str) -> int:
    """
    Compte le nombre de parenthèses ouvrantes dans une chaîne de caractères
    txt.
    """
    ouvrantes = "([{"
    nb = 0
    for c in txt:
        if c in ouvrantes:
            nb += 1
    return nb

```

Q 9 On admet que l'on a aussi écrit une fonction `compte_fermante(txt : str) -> int` qui renvoie le nombre de parenthèses fermantes dans `txt` *(On ne demande donc pas de l'écrire...)*
En utilisant les deux fonctions précédentes, écrire une fonction `bon_compte(txt: str)->bool` qui prend en paramètre une chaîne de caractères `txt` et qui renvoie `True` si `txt` a autant de parenthèses ouvrantes que parenthèses fermantes et `False` sinon.

▶ Réponse :

```
def bon_compte(txt : str) -> bool:
    """
    Vérifie si une chaîne de caractères a autant de parenthèses ouvrantes
    que fermantes.
    """
    return compte_ouvrante(txt) == compte_fermante(txt)

```

Q 10 Donner un exemple de chaîne de caractères pour laquelle `bon_compte` renvoie `True` alors qu'elle n'est pas bien parenthésée.

▶ Réponse :

```
probleme = '[2*(i+1)-3)' ou probleme = '[2*(i+1]-3)'
```

Partie B : une condition nécessaire et suffisante

Comme on vient de le voir, l'algorithme précédent n'est pas suffisant. On se propose ici d'implémenter un algorithme qui vérifie si une expression est bien parenthésée.

Q 11 Voici un algorithme écrit en langage naturel pour vérifier si une expression est bien parenthésée (on admettra qu'il est correct.) :

Algorithme proposé

- créer une liste vide nommée `prt` ;
- parcourir l'expression en testant chaque caractère :
  - si c'est une parenthèse ouvrante, on l'ajoute à la fin de la liste `prt` ;
  - si c'est une parenthèse fermante, on enlève le dernier élément de la liste `prt` et on le compare au caractère courant :
    - si les deux caractères correspondent à un couple de parenthèses, on continue le parcours ;
    - sinon l'expression n'est pas bien parenthésée ;
  - sinon on continue le parcours.
- Si l'expression a été entièrement parcourue, on teste la liste ; l'expression est bien parenthésée si la

liste est vide.

Q 12 Écrire une fonction `correspond(o: str, f: str) -> bool` qui reçoit deux caractères `o` et `f` qui renvoie `True` si `o` et `f` correspondent à un couple de parenthèses et `False` sinon.
(On pourra utiliser la méthode `index`...)

▶ **Réponse :**

```
def correspond(o,f):
    """
    Vérifie si les parenthèses ouvrantes et fermantes o et f correspondent.
    """
    ouvrantes = "([{"
    fermantes = ")]}"
    return ouvrantes.index(o) == fermantes.index(f)
```

Q 13 Écrire une fonction `est_bien_parenthesee(txt: str)->bool` qui prend en paramètre une chaîne de caractères et qui implémente l'algorithme précédent.

▶ **Réponse :**

```
def est_bien_parenthesee(txt : str) -> bool:
    """
    Vérifie si une chaîne de caractères est bien parenthésée.
    """
    ouvrantes = "([{"
    fermantes = ")]}"
    prt = []
    for c in txt:
        if c in ouvrantes:
            prt.append(c)
        elif c in fermantes:
            if len(prt) == 0: # si la liste est vide
                return False
            o = prt.pop()
            if not correspond(o,c) :
                return False
    if len(prt) == 0 :
        return True
    return False
```