ITC – DS1

I. Petits exercices sur les tableaux

Exercice 1 Écrire une fonction `pairs(t)` qui prends en entrée un tableau d'entiers et retourne le nombre d'entiers pairs y qui apparaîssent. On rappelle que le reste de la division de `a` par `b` se note `a % b`.

```
>>> pairs([4, 1, 2, 6, 3, 4, 8, 7, 6])
6
```

Exercice 2 Écrire une fonction `maximum(t)` qui, étant donné un tableau *non vide* `t` d'entiers (pas nécessairement positifs), retourne la plus grand valeur présente dans le tableau. On n'utilisera bien sûr pas la fonction `max`.

```
>>> maximum([4, 2, 3, 1, 7, 6, 3, 5, -2])
7
```

Exercice 3 Écrire une fonction `liste_records(t)` qui, étant donné un tableau `t` d'entiers passés en arguments, retourne la liste des positions des *records*, un record étant une valeur strictement supérieures à toutes les valeurs précédentes.

On pourra distinguer explicitement le cas où `t` est vide.

```
>>> liste_records([3, 1, 2, 6, 7, 5, 7, 9, 8])
[0, 3, 4, 7]
```

II. Dictionnaires

On rappelle quelques éléments d'utilisation de dictionnaires :

- ⋆ Les dictionnaires sont constitués d'un ensemble d'associations clé/valeur.
- ⋆ Le dictionnaire vide est noté `{}`.
- ⋆ La valeur associée à une clé est obtenue en écrivant `dico[clé]`, et on peut créer ou modifier une association en écrivant `dico[clé] = valeur`.
- ⋆ On teste si une clé est présente dans un dictionnaire en écrivant `clé in dico`.

Exercice 4 Écrire une fonction `comptage(t)` qui prends en entrée un tableau de chaînes de caractères, et renvoie un dictionnaire indiquant, pour chaque chaîne de caractères apparaissant dans `t`, le nombre de fois qu'elle apparait. On prendra soin de distinguer les cas suivant qu'une chaîne a déjà été rencontrée ou non.

```
>>> comptage(["a", "b", "a", "c", "b", "a"])
{"a": 3, "b": 2, "c": 1}
```

On peut parcourir un dictionnaire par ses clés en écrivant `for cle in dico: ...`. Ainsi, le code suivant affiche toutes les associations d'un dictionnaire :

```
for cle in dico:
    valeur = dico[cle]
    print(cle, valeur)
```

Exercice 5 Écrire une fonction `inversion(d)` qui a le comportement suivant : elle reçoit en entrée un dictionnaire `d` dont chaque valeur associée à une clé est une liste de chaîne de caractères, et on veut retournée le dictionnaire qui associe à chaque chaîne de caractère rencontrée l'ensemble des clés où elle apparaît dans `d`. On veut par exemple :

```
>>> inversion({1: ["Alice", "Bob"],
               3: ["Alice", "Eve"],
               4: ["Bob", "Eve", "Charles"]})
{'Alice': [1, 3], 'Bob': [1, 4], 'Eve': [3, 4], 'Charles': [4]}
```

III. 2048

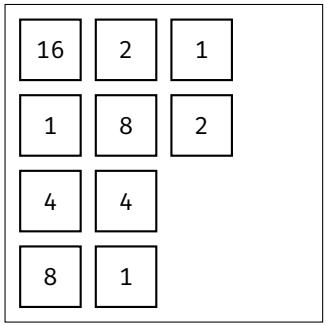
Le jeu « 2048 » est un jeu vidéo de type puzzle développé par Gabriele Cirulli et publié sous license libre en 2014.

Il se joue sur un damier de 4 × 4 cases, chaque case pouvant être vide ou comporter une valeur qui est une puissance de 2. En voici un exemple :

16	1	1	1
1		8	2
2	2	2	2
4		4	1

À chaque coup, on effectue un mouvement dans l'une des quatre directions *gauche*, *droite*, *haut* ou *bas*. À chaque fois, les cases vont le plus loin possible dans la direction indiquée, et deux cases de même valeur contigües fusionnent en une case de valeur double, puis une case vide est remplie aléatoirement.

Voici par exemple le résultat en faisant un déplacement vers la gauche à partir de la grille précédente :



Sur la ligne la plus basse, les deux 4 ont fusionné en un 8. Au-dessus, chaque paire de 2 contigüs donne deux 4. Sur la ligne la plus haute, en allant de gauche à droite, les deux premiers 1 donnent un 2 et la troisième 1 demeure inchangé.

1. Déplacement à gauche

C'est cette transformation, un déplacement vers la gauche, que nous allons essayer de programmer.

Dans la suite, une ligne de la grille sera représentée par un tableau d'entiers, les cases vides étant représentées par la valeur zéro. Ainsi, la deuxième ligne de la grille initiale, en partant du haut, est représentée par `[1, 0, 8, 2]`.

Question 6 Écrire une fonction `supprimer_zeros(t)` qui prends en entrée un tableau `t` d'entiers et renvoie un nouveau tableau obtenu à partir du précédent en supprimant tous les zéros et en conservant les autres valeurs dans le même ordre.

Le tableau retourné pourra ne pas avoir la même taille que le tableau initial.

On veut par exemple :

```
>>> supprimer_zeros([1, 0, 8, 0, 0, 2, 4])
[1, 8, 2, 4]
```

Question 7 Écrire une fonction `fusionner_a_gauche(t)` qui prends en entrée un tableau d'entier sans zéros et, en le parcourant de gauche à droite, fusionne deux entiers consécutifs égaux en un entier double.

On veut par exemple :

```
>>> fusionner_a_gauche([5, 3, 1, 2, 9])
[5, 3, 1, 2, 9]
>>> fusionner_a_gauche([4, 4, 1])
[8, 1]
>>> fusionner_a_gauche([2, 2, 2, 2])
[4, 4]
>>> fusionner_a_gauche([16, 1, 1, 1])
[16, 2, 1]
```

Question 8 Écrire une fonction `ajouter_zeros_a_droite(t, n)` qui prends en entrée un tableau `t` d'entiers et un entier `n`, et qui renvoie un tableau obtenu à partir de `t` en ajoutant des zéros à droite pour obtenir un tableau de longueur `n`. On suppose qu'au départ, `len(t)` $\leq$ `n`.

```
>>> ajouter_zeros_a_droite([2, 5, 1], 5)
[2, 5, 1, 0, 0]
```

Question 9 Écrire une fonction `deplacement_a_gauche(t)` qui prends en entrée un tableau `t` d'entiers et retourne un tableau, *de même taille que* `t`, contenant le résultat d'un déplacement à gauche, en serrant toutes les cases non vides à gauche et fusionnant les cases voisines de même valeur.

La longueur du tableau passé en argument n'est pas nécessairement égale à 4.

Par exemple,

```
>>> deplacement_a_gauche([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
[5, 4, 2, 3, 2, 0, 0, 0, 0, 0]
```

2. Détermination aléatoire d'une case vide

Nous allons maintenant simuler le processus de remplissage d'une case aléatoire.

Question 10 Écrire une fonction `liste_cases_vides(t)` qui, étant donné un tableau `t` d'entiers, retourne la liste des indices des cases vides, c'est-à-dire celles contenant la valeur `0`.

Par exemple,

```
>>> liste_cases_vides([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
[2, 4, 5]
```

Pour sélectionner aléatoirement une case vide, nous allons utiliser la bibliothèque `random` et plus particulièrement la fonction `randint` dont l'aide est la suivante :

```
>>> help(random.randint)
randint(a, b) method of random.Random instance
    Return random integer in range [a, b],
    including both end points.
```

Question 11 Quel instruction permet d'utiliser cette fonction, et comment peut-on l'utiliser ensuite ?

Question 12 Écrire une fonction `element_dans_tableau(t)` qui, étant donné un tableau `t` *supposé non vide*, retourne un élément de `t` tiré aléatoirement. Tous les éléments de `t` doivent pouvoir être choisis.

Question 13 En déduire une fonction `case_vide_aleatoire(t)` qui, étant donné un tableau `t` d'entiers, renvoie l'indice d'un case vide choisi aléatoirement. Si `t` ne contient aucune case vide, on retournera `-1`.

```
>>> case_vide_aleatoire([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
5
>>> case_vide_aleatoire([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
5
>>> case_vide_aleatoire([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
2
>>> case_vide_aleatoire([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
4
>>> case_vide_aleatoire([5, 2, 0, 2, 0, 0, 2, 3, 1, 1])
2
>>> case_vide_aleatoire([5, 2, 3, 1, 1])
-1
```

3. Bonus pour les plus rapides

Question 14 Écrire une fonction `deplacement_a_gauche(t)` qui effectue toutes les transformations nécessaire *en place*, c'est-à-dire en modifiant `t` et sans allouer de tableau supplémentaire. La fonction ne renverra rien, `t` étant modifié.

```
>>> t = [5, 2, 0, 2, 0, 0, 2, 3, 1, 1]
>>> deplacement_a_gauche(t)
>>> t
[5, 4, 2, 3, 2, 0, 0, 0, 0, 0]
```