Révisions

Exercice 1 - Fonctions

1) Écrire une fonction `double` qui, recevant un nombre, renvoie son double.

```
>>> double(5)
10
```

2) Écrire une fonction `est_pair` qui, recevant un entier, renvoie `True` s'il est pair et `False` sinon.

```
>>> est_pair(6)
True
>>> est_pair(5)
False
```

3) Écrire une fonction `somme_des_multiples` qui, recevant trois entiers `a`, `b` et `n`, renvoie la somme des multiples de `n` compris entre `a` et `b`.

```
>>> somme_des_multiples(10, 21, 3)   # 12 + 15 + 18 + 21 = 66
66
```

Exercice 2 - Listes

1) Écrire une fonction `somme` qui, recevant une liste de nombres, renvoie sa somme.

```
>>> somme([4, 6, -2, 4, 3])
15
```

2) Écrire une fonction `nb_occurrences` qui, recevant une liste `L` et un objet `x`, renvoie le nombre de fois que `x` apparaît dans `L`.

```
>>> nb_occurrences([4, 6, -2, 4, 3], 4)
2
```

3) Écrire une fonction `maximum` qui, recevant une liste de nombres, renvoie son plus grand élément.

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

4) Écrire une fonction `tous_pairs` qui, recevant une liste d'entiers, renvoie `True` si tous ses éléments sont pairs et `False` sinon.

```
>>> tous_pairs([8, 0, 4, 2])
True
>>> tous_pairs([8, 0, 5, 2])
False
```

5) Écrire une fonction `compris_entre` qui, recevant une liste de nombres `L` et deux nombres `a` et `b`, renvoie la liste des éléments de `L` compris entre `a` et `b` (au sens large).

```
>>> compris_entre([3, 5, 8, 2, -3, 6, 8, 12, 5], 3, 7)
[3, 5, 6, 5]
```

6) Écrire une fonction `multiples` qui, recevant trois entiers `a`, `b` et `n`, renvoie la liste des multiples de `n` compris entre `a` et `b`.

```
>>> multiples(10, 21, 3)
[12, 15, 18, 21]
```

Exercice 3 - Chaînes de caractères

1) Écrire une fonction `mot_le_plus_long` qui, recevant une liste de chaînes de caractères, renvoie la plus longue de ces chaînes.

```
>>> mot_le_plus_long(['Anna', 'Tom', 'Pierre', 'Julie', 'Simon'])
'Pierre'
```

2) Écrire une fonction `enchevetrer` qui, recevant deux chaînes de caractères `mot1` et `mot2`, renvoie la chaîne formée du premier caractère de `mot1`, puis du premier caractère de `mot2`, puis du deuxième caractère de `mot1`, etc.

```
>>> enchevetrer('abcd', 'efgh')
'aebfcgdh'
```

3) Écrire une fonction `doublons` qui, recevant une chaîne de caractères, renvoie la liste des caractères qui apparaissent au moins deux fois dans la chaîne.

```
>>> doublons('mathematiques')
['m', 'a', 't', 'e']
```

Exercice 4 - Dictionnaires

1) Écrire une fonction `selection` qui, recevant un dictionnaire associant à chaque article d'un magasin son prix et deux nombres `a` et `b`, renvoie la liste des articles dont le prix est compris entre `a` et `b`.

```
>>> prix = {'livre': 10, 'crayon': 1, 'jeu': 30, 'TV': 100, 'PC': 500}
>>> selection(prix, 10, 50)
['livre', 'jeu']
```

2) Écrire une fonction `fusion` qui, recevant deux dictionnaires `d1` et `d2`, renvoie un dictionnaire contenant les éléments de `d1` et `d2`.

```
>>> d1 = {'Nom': 'Einstein', 'Prénom': 'Albert', 'Age': 146}
>>> d2 = {'Profession': 'Physicien', 'Nationalité': 'Allemande'}
>>> fusion(d1, d2)
{'Nom': 'Einstein', 'Prénom': 'Albert', 'Age': 146, 'Profession':
 'Physicien', 'Nationalité': 'Allemande'}
```

Exercice 5 - Tableaux bidimensionnels

Dans cet exercice on appelle tableau une liste de listes de même longueur. Par exemple, le tableau $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ est représenté par la liste `[[1, 2, 3], [4, 5, 6]]`.

On notera que si `t` est un tableau représenté de cette manière, alors `t[i]` est la $\texttt{i}^e$ ligne de `t` et donc `t[i][j]` est le $\texttt{j}^e$ terme de la $\texttt{i}^e$ ligne. Ainsi, si `t` est le tableau ci-dessus, alors `t[0][0]` vaut `1`, `t[1][0]` vaut `4`, `t[0][1]` vaut `2`, etc.

1) Écrire une fonction `somme_tableau` qui, recevant un tableau de nombres, renvoie la somme de ses éléments.

```
>>> somme_tableau([[1, 2, 3], [4, 5, 6]])
21
```

2) Écrire une fonction `carre_tableau` qui, recevant un tableau de nombres, renvoie le tableau des carrés de ses éléments.

```
>>> carre_tableau([[1, 2, 3], [4, 5, 6]])
[[1, 4, 9], [16, 25, 36]]
```

3) Écrire une fonction `transposee` qui, recevant une matrice représentée par un tableau de nombres, renvoie sa transposée.

```
>>> transposee([[1, 2, 3], [4, 5, 6]])
[[1, 4], [2, 5], [3, 6]]
```

Exercice 6 - Représentation des entiers

1) Convertir 92 en base 2, puis écrire -92 sur 8 bits avec la méthode du complément à 2.

2) Convertir $\overline{101110}_2$ en base 10.

Exercice 7 - Récursivité

Les nombres de Catalan sont les entiers définis par $C_0 = 1$ et par la relation de récurrence :

$$\forall\, n \in \mathbb{N}, C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}.$$

1) Calculer à la main C_n pour $n \in \{1, \ldots, 5\}$.

2) Écrire une fonction récursive `catalan` qui, recevant un entier naturel n, renvoie la valeur de C_n.

```
>>> catalan(5)
42
```

Pour que la fonction soit utilisable avec de grandes valeurs de n il faudra utiliser la mémoïsation.

```
>>> catalan(100)  # doit être instantané
896519947090131496687170070074100632420837521538745909320
```

Exercice 8 - Graphes

1) Écrire une fonction `nombre_aretes` qui, recevant un graphe non orienté sans boucle représenté par un dictionnaire d'adjacence, renvoie le nombre de ses arêtes.

```
>>> graphe = {'A': ['B', 'C'],
              'B': ['A', 'D', 'E'],
              'C': ['A', 'F'],
              'D': ['B'],
              'E': ['B', 'F'],
              'F': ['C', 'E']}
>>> nombre_aretes(graphe)
6
```

2) Le complémentaire d'un graphe G est le graphe ayant les mêmes sommets que G et tel que deux sommets sont reliés par une arête si et seulement si ils ne le sont pas dans G. Écrire une fonction `complementaire` qui, recevant un graphe représenté par un dictionnaire d'adjacence, renvoie son complémentaire.

```
>>> complementaire(graphe)
{'A': ['D', 'E', 'F'], 'B': ['C', 'F'], 'C': ['B', 'D', 'E'],
 'D': ['A', 'C', 'E', 'F'], 'E': ['A', 'C', 'D'], 'F': ['A', 'B', 'D']}
```

3) Reprendre les questions précédentes lorsque les graphes sont représentés par des matrices d'adjacence.

```
matrice = [[0, 1, 1, 0, 0, 0],
           [1, 0, 0, 1, 1, 0],
           [1, 0, 0, 0, 0, 1],
           [0, 1, 0, 0, 0, 0],
           [0, 1, 0, 0, 0, 1],
           [0, 0, 1, 0, 1, 0]]
>>> nombre_aretes(matrice)
6
>>> complementaire(matrice)
[[0, 0, 0, 1, 1, 1], [0, 0, 1, 0, 0, 1], [0, 1, 0, 1, 1, 0],
 [1, 0, 1, 0, 1, 1], [1, 0, 1, 1, 0, 0], [1, 1, 0, 1, 0, 0]]
```