TP 7
Révisions et consolidation 1

Exercice 1 *Solide sur les bases*

1. Écrire une fonction `pH(x)` qui prend en argument un nombre x (on suppose que x est entre 0 et 14 et représente bien le potentiel hydrogène d'une solution) et affiche le mot « acide », « basique » ou « neutre », en fonction du pH. Éventuellement, la fonction affiche « invalide » si x n'est pas dans $[0, 14]$.
2. Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \frac{8}{2+u_n}$. Écrire des fonctions, indépendantes l'une de l'autre :

   (i) `u_suite(n)` : renvoie le terme u_n,

   (ii) `u_liste(n)` : renvoie la liste des n premiers termes de la suite.
3. Écrire une fonction `somme(n)` qui prend en argument un nombre entier n et qui renvoie la valeur de la somme $\sum_{k=1}^{n} \frac{1}{k^3}$, c'est-à-dire $1 + \frac{1}{2^3} + \frac{1}{3^3} + \cdots + \frac{1}{n^3}$, et tester pour des valeurs de n de plus en plus grandes.

Exercice 2 *Suites*

La **suite de Tribonacci** est la suite $(T_n)_{n\in\mathbb{N}}$ définie par

$$T_0 = 0,\ T_1 = 1,\ T_2 = 1 \quad \text{et} \quad \forall n \in \mathbb{N},\ T_{n+3} = T_{n+2} + T_{n+1} + T_n$$

Écrire séparément les fonctions (sans que l'une ne fasse appel à l'autre) :

1. `tribonacci(n)` : calcule le terme T_n.
2. `tribonacci_liste(n)` : renvoie la liste des n premiers termes de la suite.

Pour tester on pourra vérifier que les premiers termes de la suite sont :

n	0	1	2	3	4	5	6	7	8	9	10
T_n	0	1	1	2	4	7	13	24	44	81	149

Exercice 3 *Modélisation*

On représente un nombre complexe par une liste de longueur 2 formée de sa partie réelle et de sa partie imaginaire. Par exemple le nombre complexe $z = 3 - 4i$ correspondra à la variable `z = [3, -4]`. Un nombre réel a est identifié avec la liste `[a, 0]`, et le nombre i à `[0, 1]`.

1. Quelle syntaxe permet d'obtenir la partie réelle du nombre représenté par la liste `z` ? Et la partie imaginaire ?
2. Écrire une fonction `somme(z, w)` qui prend en argument deux listes `z`, `w`, représentant des nombres complexes z, w, et qui renvoie une liste représentant la somme $z + w$.
3. Écrire une fonction `produit(z, w)` qui prend en argument deux listes `z`, `w`, représentant des nombres complexes z, w, et qui renvoie une liste représentant le produit de nombres complexes $z \times w$.
4. En utilisant la fonction précédente, écrire une fonction `puissance(z, n)` qui prend en argument une liste `z` représentant un nombre complexe z, et un entier n (supposé positif), et renvoie une liste représentant le nombre complexe z^n.

   On pourra tester avec les puissances successives de $z = 1 + 2i$, qui sont :

n	0	1	2	3	4	5	6
$(1+2i)^n$	1	$1+2i$	$-3+4i$	$-11-2i$	$-7-24i$	$41-38i$	$117+44i$

Exercice 4 *Compter*

On souhaite se donner un nombre réel $r \geqslant 0$ et compter le nombre de couples d'entiers $(n, m) \in \mathbb{Z}^2$ tels que $n^2 + m^2 \leqslant r^2$. Ce sont les points à coordonnées entières situés à l'intérieur du cercle de rayon r.

1. Pourquoi peut-on supposer $-r \leqslant n \leqslant r$ et $-r \leqslant m \leqslant r$?
2. Écrire une fonction `compte_points(r)` qui prend en argument un nombre entier r supposé positif et qui compte le nombre de tels couples.
3. Modifier la fonction pour renvoyer non pas le nombre N de points mais le quotient N/r^2, et tester avec des valeurs de r de plus en plus grandes, par exemple 10, 100, 1000 (puis augmenter progressivement par multiples de 1000, sans dépasser 10 000). Qu'en pensez-vous ?

Exercice 5 *Tests sur les listes*

Toutes les fonctions de cet exercice prennent en argument une liste `L`, qu'on suppose constituée uniquement de nombres entiers.

1. Écrire une fonction `tous_chiffres(L)` qui renvoie `True` si tous les nombres dans `L` sont entre 0 et 9 (au sens large), et `False` sinon.
2. Écrire une fonction `sont_consécutifs(L)` qui renvoie `True` si `L` est constituée d'une suite de nombres consécutifs (par exemple `L = [13, 14, 15, 16]`) et `False` sinon.
3. Écrire une fonction `est_symétrique(L)` qui renvoie `True` si la liste `L` peut se lire aussi bien à l'envers qu'à l'endroit, par exemple `L = [4, 5, 8, 2, 8, 5, 4]`.

Exercice 6 *Création de listes*

1. Pour une liste `L`, la *liste miroir* de `L` est la liste rangée en ordre inverse. Par exemple pour `L = [3, 7, 2, 8]` c'est la liste `[8, 2, 7, 3]`. Écrire une fonction `miroir(L)` qui renvoie la liste miroir de `L`.
2. Pour une liste `L` de longueur n, la *liste des sommes cumulées* de `L` est la liste `C` de longueur n où `C[i]` est la somme de tous les termes de `L` d'indice inférieur à i. Par exemple la liste des sommes cumulées de `[2, 4, 3, 5, 3, 5]` est `[2, 6, 9, 14, 17, 22]`. Remarquez que le premier terme est toujours `L[0]` et le dernier est la somme de tous les termes de `L`.

   Écrire une fonction `sommes_cumul(L)` qui renvoie la liste des sommes cumulées de `L`.
3. Écrire une fonction `somme(L, M)` qui prend en argument deux listes supposées de même longueur et qui renvoie la liste de leurs sommes terme à terme (la liste des `L[i] + M[i]`). Par exemple la somme de `L = [3, 7, 1]` et `M = [2, 8, 1]` est `[5, 15, 2]`.

Exercice 7 (*) *Alphabet*

On donne la chaine de caractères suivante : `alphabet = "abcdefghijklmnopqrstuvwxyz"`. Ainsi on considère que la lettre `a` est le caractère numéro 0 de l'alphabet, et `z` est le caractère numéro 25.

1. Écrire une fonction `numéro(x)` qui prend en argument un caractère seul `x` (on suppose que c'est l'une des 26 lettres de l'alphabet ci-dessus : pas de majuscule, pas d'accents) et qui renvoie le numéro de `x` en tant que lettre de l'alphabet. Si `x` n'est pas une lettre de l'alphabet on pourra renvoyer l'objet spécial `None`.
2. Écrire une fonction `compte_lettres(s)` qui prend en argument une chaine de caractères `s` et qui renvoie une liste `C` de longueur 26, où `C[x]` indique combien de fois apparait la lettre numéro `x` dans la chaine `s`.

L'une des méthodes les plus anciennes pour coder un texte en un message secret s'appelle *codage de César*, utilisée effectivement par Jules César, et consiste à remplacer chaque lettre d'un texte par celle trois lettres plus loin dans l'alphabet. Ainsi `a` est remplacé par `d`, `b` est remplacé par `e`, etc, et `x` est remplacé par `a`, `y` par `b` et enfin `z` par `c`.

3. Écrire une fonction `code_caractere(x)` qui prend en argument un caractère `x` et qui renvoie le caractère codé par ce procédé.

4. Écrire une fonction `code(s)` qui prend en argument une chaine de caractères `s` et renvoie la chaine ainsi codée. On pourra pour cela passer par la création d'une *liste* des caractères codés. Après avoir créé une telle liste `L`, renvoyer non pas `L` mais `"".join(L)` qui colle tous les caractères en une seule chaine.

   On pourra dans un premier temps supposer que `s` contient seulement des lettres de l'alphabet parmi ces 26 là — mais on pourra lever cette restriction en ne « codant pas » les autres caractères (notamment les espaces).