TP 7
Révisions et consolidation

Exercice 1. *Suites*
La *suite de Tribonacci* est la suite $(T_n)_{n\in\mathbb{N}}$ définie par la relation de récurrence

$$\forall n \in \mathbb{N}, \quad T_{n+3} = T_{n+2} + T_{n+1} + T_n \tag{1}$$

avec les conditions initiales $T_0 = 0$, $T_1 = 1$, $T_2 = 1$.
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
T_n	0	1	1	2	4	7	13	24	44

(2)

Exercice 2. *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 $z = 1 + i$, qui sont

n	0	1	2	3	4	5	6
$(1+i)^n$	1	$1+i$	$2i$	$-2+2i$	-4	$-4-4i$	$-8i$

(3)

Exercice 3. *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 4. *Sommes*
La fonction arctangente $\arctan(x)$ est assez bien approchée, quand x est petit, par la somme

$$\arctan(x) \approx \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{2k+1} = 1 - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots + (-1)^n \frac{x^{2n+1}}{2n+1} \tag{4}$$

L'approximation est d'autant meilleure que n est grand et que x est petit.

1. Écrire une fonction `arctan(x, n)` qui calcule cette somme.
2. Sachant que $\frac{\pi}{4} = \arctan(1)$, en déduire une fonction `pi_arctan(n)` qui calcule le nombre π avec la formule ci-dessus.

3. Sachant que $\frac{\pi}{4} = 4\arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$, en déduire une autre fonction `pi_arctan_machin(n)` qui calcule le nombre π.
4. Pour tester les deux fonctions précédentes, on pourra essayer avec des valeurs de n de plus en plus grandes et afficher la valeur absolue de l'écart avec le nombre π, qu'on trouve en tant que constante `pi` dans le module `math`.
   Afficher les écarts à π pour chacune des deux fonctions, avec des petites valeurs de n au départ (par exemple 5, 10, 15, 20), puis avec des grandes valeurs (100, 1000, 10 000, et même 100 000 et 1 000 000).

Exercice 5. *Listes*

1. Écrire une fonction `moyenne(L)` qui calcule la moyenne des valeurs de la liste `L`.
   Attention à la liste vide ! C'est peut-être une occasion d'utiliser **`assert`** *(TP 3, § V).*
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.

Exercice 6. *Algorithmes sur les listes*

Une *permutation de longueur n* est une liste de longueur n où chacun des nombres de 0 à $n-1$ apparait exactement une fois. Par exemple `[3, 1, 0, 2]` est bien une permutation de longueur 4.

1. Pourquoi suffit-il que chacun de ces nombres apparaisse *au moins* une fois ? Ou bien *au plus* une fois ?
2. Écrire une fonction `appartient(L, x)` qui renvoie **`True`** si le nombre `x` est présent dans la liste `L` et **`False`** sinon.
3. En utilisant la fonction précédente, écrire une fonction `est_permutation(L)` qui renvoie **`True`** si `L` est bien une permutation, et **`False`** sinon.

Une autre possibilité plus directe est la suivante. Pour tester si la liste `L` est bien une permutation, on crée une liste de booléens `M` de même taille que `L`, et on parcourt une seule fois `L`, mais on « coche » les nombres qu'on a vu. Ainsi `M[x] = True` est à interpréter comme « `x` est bien présent dans `L` » alors que `M[x] = False` signifie que `x` n'a pas encore été rencontré.

4. En utilisant cette méthode, écrire une fonction `est_permutation_2(L)`.

Exercice 7. *Texte*

Pour un caractère seul `x`, la méthode `x.isalpha()` renvoie **`True`** si `x` est un caractère alphabétique (une lettre, accentuée ou non, minuscule ou majuscule) et **`False`** sinon (espace, signe de ponctuation, chiffre, etc).

1. Écrire une fonction `compte_lettres(s)` qui prend en argument une chaine de caractères `s` et compte le nombre de caractères alphabétiques que contient `s`.
2. Écrire une fonction `garde_lettres(s)` qui prend en argument une chaine de caractères `s` et renvoie la liste formée de tous les caractères alphabétiques de `s`.
3. Dans la fonction ci-dessus renvoyer non pas la liste `L` mais `"".join(L)`, qui colle tous les caractères collectés en une seule chaine.

Exercice 8. *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 `numero(x)` qui prend en argument un caractèr 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 liste de ses caractères ainsi codés.
   Éventuellement, on pourra 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).
   Enfin, après avoir créé une liste `L` de caractères codés, on peut renvoyer `"".join(L)` pour re-transformer la liste en une chaine de caractères.