TP N°1: Rappel général

Objectif de ce TP: se rappeler des principales notions de base du langage de programmation `Python` vues en 1ère année.
Avant de commencer: Créez un dossier *TP1* dans «`D:\votre_groupe\votre_nom_prenom`» dans lequel vous placerez tous vos scripts que vous créerez durant ce TP.

Exercice 1: Crible d'Eratosthène

Un nombre premier est un entier qui n'est divisible que par 1 et par lui-même. On se propose d'écrire un programme qui établit la liste de tous les nombres premiers compris entre 1 et 100, en utilisant la méthode du *crible d'Eratosthène* :

- Créer une liste de 100 éléments, chacun initialisé à la valeur 1 ;
- Parcourir cette liste à partir de l'élément d'indice 2: si l'élément analysé possède la valeur une, mettez à zéro tous les autres éléments de la liste, dont les indices sont les multiples de l'indice auquel vous êtes arrivé.

Lorsque vous aurez parcouru ainsi toute la liste, les indices des éléments qui seront restés à 1 seront les nombres premiers recherchés.

En effet, à partir de l'indice 2, vous annulez tous les éléments d'indices pairs $4, 6, 8, \ldots$
Avec l'indice 3, vous annulez les éléments d'indices $6, 9, \ldots$
Et ainsi de suite. Seul resteront à 1 les éléments dont les indices sont effectivement des nombres premiers.

Travail à faire :

1. Écrire une fonction `init(L)` permettant d'initialiser à 1 une liste `L` de 100 éléments .
2. Écrire une fonction `multiple(L, i)` permettant de mettre à zéro les éléments dont l'indice est un multiple d'un entier donné comme paramètre.
3. Écrire une fonction `suivant(L, i)` qui à partir d'un indice donné, retourne le premier indice dont l'élément correspondant est différent de zéro.
4. Écrire une fonction `crible(L)` permettant d'appliquer la méthode présentée par la crible d'Eratosthène pour retourner une liste qui contient les nombres premiers (il suffit d'étirer jusqu'à la racine entière de 100).
5. Écrire une fonction récursive `crible_recursive` permettant d'appliquer la méthode présentée pour retourner la liste initiale après les modifications.
6. Donner le schéma d'exécution de la fonction précédente (si on veut déterminer les entiers premiers qui sont inférieur à 10).
7. Écrire un programme qui fait appel aux fonctions déjà définies afin d'afficher les entiers premiers compris entre 1 et 100.

Exercice 2: fonction passée en argument

Écrire les fonctions Python suivantes :

1. `carre(n)` : calcule et retourne le carré d'un entier n passé en paramètre.
2. `somme_fonction(f, n)` : calcule et retourne la somme $\sum_{i=0}^{n} f(i)$ pour une fonction f qui retourne le carré du paramètre n=10.

Exercice 3: Recherche dichotomique

On suppose que `L` est une liste triée de réels distincts et que x est un réel vérifiant $L[0] \leq x < L[n-1]$ où n est la taille de la liste.
Écrire une fonction `dicho(x,L)` retournant l'unique entier i vérifiant $L[i] \leq x < L[i+1]$ en faisant une recherche dichotomique.

Exercice 4: Suite de polynômes

On définit une suite de polynôme SP par $\begin{cases} SP_0(x) & = P(x) \\ SP_1(x) & = -P^{'}(x) \\ SP_{k+2}(x) & = Q(x) * SP_{k+1}(x) + SP_k(x) \end{cases}$

Avec :

- $P(x)$ un polynôme de degré n $(n \neq 0)$ tel que : $P(x) = \sum_{i=0}^{n} a_i x^i$
- $P^{'}(x)$ le polynôme dérivé de degré $n-1$ tel que : $P^{'}(x) = \sum_{i=0}^{n-1} (i+1) a_{i+1} x^i$
- $Q(x)$ est le produit des deux polynômes $SP_{k+1}(x)$ et $SP_k(x)$.

Tout polynôme de degré n sera représenté sous forme d'une liste de taille $n+1$ tel que les coefficients sont les éléments et les degrés sont les indices.

Exemple : La liste correspondante au polynôme $P(x) = 1 + x - 5x^2 + x^3 - 2x^4 + 6x^6$ est : `L=[1,1,-5,1,-2,0,6]`

On désire déterminer à partir de la suite ci-dessus le polynôme $SP_m(x)$ avec $m \geq 2$.

Travail à faire :

1. Écrire une fonction appelée `saisie_deg()` qui permet de saisir un entier strictement positif. Ajouter un bloc `try-except` qui gère une exception de type `ValueError`, ce qui pourrait se produire si l'utilisateur saisie une valeur non numérique. Imprimer un message d'erreur si cela se produit.
2. Écrire une fonction appelée `saisie_poly(n)` permettant de saisir la représentation d'un polynôme $P(x)$ de degré n dans une liste.
3. Écrire une fonction appelée `derive(P)` permettant de déterminer la représentation du polynôme dérivé d'un polynôme P représenté par une liste, le résultat sera une liste.
4. Écrire une fonction appelée `opp_poly(P)`permettant de déterminer la représentation du polynôme opposé d'un polynôme P représenté par une liste, le résultat sera une liste.
5. Écrire une fonction appelée `add_poly(P1,P2)` qui prend comme paramètres deux polynômes $P1$ et $P2$ et retourne la représentation du polynôme $P1 + P2$.
6. Écrire une fonction appelée `mul_poly(P1, P2)` qui prend comme paramètres deux polynômes $P1$ et $P2$ et retourne la représentation du polynôme $P1 * P2$.
   Sachant que si $P1(x) = \sum_{i=0}^{n1} a_i x^i$ et $P2(x) = \sum_{i=0}^{n2} b_i x^i$ alors le produit est $P1(x) * P2(x) = \sum_{k=0}^{n1+n2} c_k x^k$
   avec $c_k = \sum_{i+j=k} a_i b_j$.
7. Écrire le programme principal permettant de :
   (a) Saisir le degré n d'un polynôme.
   (b) Saisir la liste qui représente le polynôme P.
   (c) Déterminer et afficher la liste relative au $i^{\text{ème}}$ terme de la suite du polynôme SP.

Exercice 5: Les fichiers

1. Écrivez une fonction `copier(fichier)` qui recopie un fichier texte. Le programme demandera à l'utilisateur le nom (avec chemin d'accès) de fichier et le recopier dans le même emplacement.
2. Écrire une fonction `comparer(f1, f2)` qui compare les contenus de deux fichiers et signale la première différence rencontrée.
3. A partir de deux fichiers préexistants `f1` et `f2`, construire un fichier `f3` qui contienne alternativement un élément de `f1`, un élément de `f2`, un élément de `f1`, et ainsi de suite, jusqu'à atteindre la fin de l'un des deux fichiers originaux. Compléter ensuite `f3` avec les éléments restants sur l'autre.