TD N°0: Rappel général

Objectif : se rappeler des principales notions de base du langage de programmation `Python` vues en 1ère année.

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 2 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 1, 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()` permettant de retourner une liste de 100 éléments initialisés à 1 .
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()` 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 des nombres premiers.
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 paramètre

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 $\sum_{i=0}^{n} f(i)$, où f est une fonction passée en paramètre.
3. Utiliser la fonction `somme_fonction` pour calculer la somme des carrés des entiers $i \in [1, 10]$.

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: Récursivité

Une chaine est palindrome si elle se lit de la même manière de gauche vers la droite et de droite vers la gauche.
Exemples : ELLE, NON, RADAR.
Écrire une fonction récursive qui permet de vérifier si une chaine donnée est palindrome

Exercice 5: 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 un polynôme de degré n $(n \neq 0)$ tel que : $P(x) = \sum_{i=0}^{n} a_i x^i$
- P' le polynôme dérivé de degré $n-1$ tel que : $P'(x) = \sum_{i=0}^{n-1} (i+1) a_{i+1} x^i$
- Q est le produit des deux polynômes SP_{k+1} et SP_k.

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 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 un polynôme P.
   (c) Déterminer et afficher la liste relative au $i^{\text{ème}}$ terme de la suite du polynôme SP.

Exercice 6: Manipulation de fichiers

Vous travaillerez avec un fichier texte qui stocke des informations sous forme de lignes, chaque ligne contenant un enregistrement (des noms et des âges séparés par une virgule).
Par exemple: `"'Ali', 20"`
Écrivez les fonctions Python suivantes :

1. `ecrire_fichier(fichier, donnees)` : écrit une liste de chaînes de caractères dans un fichier, chaque chaîne représentant un enregistrement. Si le fichier existe, son contenu est remplacé.
2. `chercher_fichier(fichier, mot_cle)` : recherche un enregistrement contenant le mot clé passé en paramètre et retourne l'enregistrement correspondant.

Exercice 7: Dictionnaire, set, tuple, fichier ...

Etant donné un ensemble de points du plan, on veut identifier le couple de points les plus proches au sens de la distance euclidienne (utile dans les transports, aériens ...). Nous représenterons chaque **Point** par un tuple de flottants (x, y) représentant ses coordonnées. L'ensemble des Points sera stocké dans une structure de type **set**.

1. Écrire une fonction **dist(p, q)** qui retourne la distance euclidienne entre deux Points **p** et **q**.
2. Écrire une fonction **plus_proches_voisins(ensPts)** qui prend pour argument l'ensemble (set) des Points et retourne un couple de Points situés à une distance minimale l'un de l'autre. Pour simplifier, il suffit de faire une recherche exhaustive (complexité $O(n^2)$ avec `n = len(ensPts)`).
3. Écrire une fonction **loadCities** permettant de récupérer les coordonnées de villes à partir d'un fichier texte où chaque ligne est de la forme : `nom_ville:abscisse:ordonnée`

   Exemple :

   `cities.txt`

```
Sousse:10.63699:35.82539
Douz:9.038601:33.456535
Sidi Bouzid:9.48494:35.03823
Sfax:10.766163:34.747847
Gabes:10.097522:33.888077
Touzeur:8.122933:33.918534
```

   Cette fonction prend en paramètres le nom du fichier et renvoie un dictionnaire `Dcities` contenant les données des villes récupérées à partir du fichier. Le dictionnaire `Dcities` a le format suivant :

   - Chaque `clé` est un *Point* représentant les coordonnées d'une ville;
   - Chaque `valeur` est une chaine (str) représentant le nom d'une ville.

4. Écrire une fonction **plus_proches_villes(Dcities)** qui prend pour argument le dictionnaire `Dcities` construit précédemment, et retourne le couple des noms des villes situées à une distance minimale l'une de l'autre. (Important : pensez à utiliser la fonction `plus_proches_voisins`)
5. Écrire la fonction **writeCities(Dcities, nomF)** qui prend en paramètre le dictionnaire `Dcities` et permet d'écrire les coordonnées des villes extraites de `Dcities` dans le fichier `nomF` où chaque ligne est de la forme `nom_ville:abscisse:ordonnée`.

ANNEXE - Quelques fonctions Python

`source.split(motif)` retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.

`f=open(nomFichier, 'r' ou 'w' ou 'a')` ouvre le fichier en mode *'r' : lecture*, *'w' : écriture* ou *'a' : ajout*.

`f.close()` permet de fermer un fichier.

`f.readlines()` permet de lire et retourner le contenu de toutes les lignes d'un fichier dans une liste

`f.write(ch)` écrit la chaîne `ch` à partir de la position courante du fichier `f`.

On rappelle les opérations utiles sur un dictionnaire `D` :

- `D[k]` retourne la valeur associée à la clef `k` du dictionnaire `D`. Si la clef `k` n'existe pas dans `D`, cette syntaxe génère une erreur.
- `D.keys()` retourne un itérable formé par les clefs du dictionnaire `D`.
- `D.values()` retourne un itérable formé par les valeurs du dictionnaire `D`.
- `D.items()` retourne un itérable formé par des couples `(k,v)` où `k` est une clef du dictionnaire `D` et `v` est la valeur associée.

On rappelle les opérations utiles sur un ensemble `S` :

- `S.add(element)` ajoute l'`element` à l'ensemble `S`. Si l'`element` existe déjà dans `S`, il ne sera pas ajouté à nouveau.
- `S.remove(element)` supprime l'`element` de l'ensemble `S`. Si l'`element` n'existe pas dans `S`, cette syntaxe génère une erreur.
- `S.discard(element)` supprime l'`element` de l'ensemble `S` sans générer d'erreur si l'`element` n'existe pas.
- `S.pop()` supprime et retourne un élément arbitraire de l'ensemble `S`. Génère une erreur si `S` est vide.
- `S.union(T)` retourne un nouvel ensemble contenant tous les éléments de `S` et `T`.
- `S.intersection(T)` retourne un nouvel ensemble contenant tous les éléments communs à `S` et `T`.
- `S.difference(T)` retourne un nouvel ensemble contenant tous les éléments de `S` qui ne sont pas dans `T`.