EEIA 2023 | Informatique

Exercices sur les fonctions et structures de données

I. Pour commencer...

1. Déclarer la liste `lst` suivante : `[14, 7, 6, 12, 2, 3, 3, 10]`. Affichez-la.

   - Ensuite, modifier la liste de telle façon que son dernier élément soit divisé par 2 et la réafficher.
   - Modifier la liste de telle façon que l'on retranche 1 à tous ses éléments et la réafficher.
   - Afficher tous les éléments de la liste sur des lignes différentes.
   - Faire afficher seulement les nombres pairs (sans en fabriquer de nouvelles).
   - Faire afficher 10 fois chaque élément de la liste sur la même ligne séparée par des espaces.
   - Afficher chaque élément autant de fois que sa valeur. Par exemple, 14 sera affiché 14 fois.

2. Écrire une fonction qui calcule la somme des éléments d'une liste de nombres réels

3. Écrire une fonction qui calcule la moyenne des éléments d'une liste de nombres réels.

4. Écrire une fonction qui calcule le maximum des éléments d'une liste non vide de nombres réels.

5. Modifier votre fonction pour qu'elle renvoie l'indice du maximum. Dans le cas où il y a plusieurs éléments maximaux, quel indice renvoyez-vous ?

6. Définir la liste `liste` suivante : `liste =[17, 38, 10, 25, 72]`. Effectuez les opérations suivantes :

   - ajoutez l'élément `12` à la liste et affichez la liste ;
   - renversez et affichez la liste ;
   - affichez l'indice de l'élément `17` ;
   - enlevez l'élément `38` et affichez la liste ;
   - affichez la sous-liste du 2e au 3e élément ;
   - affichez la sous-liste du début au 2e élément ;
   - affichez la sous-liste du 3e élément à la fin de e la liste ;
   - affichez la sous-liste complète de la liste ;
   - affichez le dernier élément en utilisant un indiçage négatif.

7. Utilisez une liste en compréhension pour obtenir la liste [**'ad', 'ae', 'bd', 'be','cd', 'ce'**] à partir des chaînes **"abc"** et **"de"**. *Indication* : utilisez deux boucles `for` imbriquées.

8. Utilisez une liste en compréhension pour calculer la somme d'une liste d'entiers de 0 à 9. Vous pouvez utiliser la fonction de la question 1.

9. Écrire une fonction `compterMots` ayant comme argument une chaîne de caractères `er` et qui renvoie un dictionnaire qui contient la fréquence de tous les mots de la chaîne entrée.

10. Écrire une fonction `est_present` prenant en entrée un motif `m` et un mot `w` (sous forme de chaînes de caractères), et testant si le motif `m` est présent dans `w`. Comme dans l'exercice 4, on renvoie une position de `m` dans `w` si elle existe, ou `None` sinon. (Bien sûr ne pas utiliser `in`) *Indication* : Les chaînes de caractères permettent d'utiliser le *slicing*, c'est-à-dire de faire des comparaisons du type `w[i:j] == m`.

```
>>> est_present ('titi', 'tructititoto')
4
>>> est_present ('tagada', 'tructititoto')
None
```

11. Reprogrammer la fonction précédente, mais en s'interdisant le *slicing*.

12. Écrire une fonction `rechercher_remplacer` qui prend en entrée un motif `m`, un mot `w` et un mot de remplacement `r` (sous forme de chaînes de caractères), et qui renvoie un mot identique à `w`, mais dans lequel la première occurrence de `m` a été remplacée par `r`. Dans le cas où `m` est absent de `w`, on renvoie `w` inchangé.

II. Entraînement...

1. Ecrire une fonction `digital_sum` qui calcule la somme des chiffres d'un nombre entier n qu'on lui donne. Vous pouvez reprendre ce qui a été fait à la question II.8 de la première feuille d'exercice.

2. Écrire une fonction `digital_root` qui calcule la somme des chiffres d'un nombre entier n qu'on lui donne jusqu'à obtenir un nombre à un seul chiffre. *Ce résultat s'appelle* ***résidu numerique*** *ou* ***racine numérique*** *du nombre* n. Par exemple `digital_root(298) = 1` car `digital_sum(298) = 19` et `digital_sum(19) = 10` et `digital_sum(10) = 1`.

3. Écrire une fonction `additive_persistence` qui alcule la **persistence additive** d'un nombre entier n qu'on lui donne, c'est-à-dire le nombre de fois où l'on doit sommer ses chiffres pour atteindre son résidu numerique. Par exemple `additive_persistence(298) = 3`.

4. **Pour les matheux :**

   - Prouver que le résidu d'un nombre est son reste modulo 9.
   - Prouver que le résidu du produit de 2 nombres premiers jumeaux est toujours 8. Utiliser la congruence modulo 9 : quels sont les restes

possibles d'un nombre premier modulo 9 ? (est-il possible d'avoir 0, 1, 3, 6 ?)

5. Ecrire une fonction qui trie par ordre croissant les éléments d'un tableau de nombres qui lui est passé en paramètre. (*Pour les plus avancés, faites en sorte que la fonction trie le tableau en $n\log(n)$. Pouvez-vous le faire en n si on vous dit que le tableau ne contient que des entiers ?*)
6. Écrire une fonction qui calcule la médiane des éléments d'une liste non vide de nombres réels.
7. Définitions :
   - On appelle nombre *premier* tout entier naturel supérieur à 1 qui possède exactement deux diviseurs, lui-même et l'unité ;
   - On appelle *diviseur propre* de n, un diviseur quelconque de n, n exclu ;
   - un entier naturel est dit *parfait* s'il est égal à la somme de tous ses diviseurs propres ;
   - les nombres a tels que : $a + n + n^2$ est premier pour tout n tel que $0 \leq n < (a - 1)$, sont appelés *nombres chanceux*.

Écrire un module `parfait_chanceux_m.py` définissant quatre fonctions : `sommeDiviseurs`, `estParfait`, `estPremier`, `estChanceux` :

- la fonction `sommeDiviseurs` retourne la somme des diviseurs propres de son argument ;
- les trois autres fonctions vérifient la propriété donnée par leur définition et retourne un booléen. Plus précisément, si par exemple la fonction `estPremier` vérifie que son argument est premier, elle retourne `True`, sinon elle retourne `False`.
- Dans un fichier de test `test.py`, testez les fonctions du module `parfait_chanceux_m.py` sur quelques exemples.

III. Toujours plus...

1. **Jour de la semaine**. Écrivez un programme qui prend une date sous la forme `dd/mm/aaaa` et renvoie le jour de la semaine correspondant. Par exemple, si le programme part de la date `19/07/2022`, il doit renvoyer `Mardi`. *Indications* :
   - On sait que le 01/01/1970 est un jeudi. (Pourquoi cette date ? parce que c'est une date très célèbre en informatique)
   - On sait qu'une année est bissextile si elle est divisible par 4... sauf si elle est divisible par 100... sauf si elle est divibile par 400. Vous suivez ?!
2. **La suite de Conway**. La suite de Conway est une suite mathématique inventée en 1986 par le mathématicien John Horton Conway. Le premier terme de la suite de Conway est posé comme égal à 1. Chaque terme

de la suite se construit en annonçant le terme précédent, c'est-à-dire en indiquant combien de fois chacun de ses chiffres se répète.

- $x_0 = 1$
- $x_1 = 11$ (le terme précédent comporte **un "1"**)
- $x_2 = 21$ (le terme précédent comporte **deux "1"**)
- $x_3 = 1211$ (le terme précédent comporte **un "2"** et **un "1"**)
- $x_4 = 111221$
- etc.

Ecrire une fonction qui prend en paramètre un entier `n` et renvoie le n^e terme de la suite de Conway.

3. **Parenthésage correcte**. Écrire une fonction qui détermine si une suite de parenthèses `(, ), [, ], {, }` qu'on lui passe est correcte, c'est-à-dire si toute parenthèse ouverte est bien fermée dans le bon ordre.
   - Exemples de parenthésages correctes : `(){}`, `{[()]}`, `()[](({}))`
   - Exemples de parenthésages incorrectes : `(]`, `({)}`