TP d'informatique n°2 : listes et boucles « for » suite

PCSI 2023 – 2024

♡ Les exercices indiqués par un cœur dans la marge sont ceux qu'il faut absolument savoir faire très rapidement et adapter en cas de besoin.

I Syntaxe plus évoluée.

- La fonction **range**`(n)` renvoie un itérateur de $[\![0,n-1]\!]$.
- La fonction **range**`(n,p)` renvoie un itérateur de $[\![n,p-1]\!]$.
- La fonction **range**`(n,p,k)` renvoie un itérateur de $[\![n,p-1]\!]$ parcouru avec un pas de `k`.

Tester par exemple : **list**`(`**range**`(10))` ; **list**`(`**range**`(3,9))` ; **list**`(`**range**`(1,15,3))` ; **list**`(`**range**`(5,-3,-2))`.

Exercice n°1 Créez une fonction qui calcule la somme des multiples de 3 plus petits qu'un entier n que l'on passera en argument à la fonction.

Exercice n°2 Écrire une fonction `inverse_liste` qui prend en argument une liste et renvoie une autre liste qui contient les mêmes éléments que la première mais « à l'envers ». Par exemple `inverse_liste([1,2,3,4])` renverra `[4,3,2,1]`. (Pleins de manière de faire cet exercice !)

Exercice n°3 Écrire une fonction `positif_un(L)` qui prend en argument une liste et renvoie `True` s'il existe un élément positif dans la liste et `False` sinon. Proposer des exemples et tester votre fonction.

♡

Exercice n°4 Écrire une fonction `positif_all(L)` qui prend en argument une liste et renvoie `True` si tous les éléments de la liste sont positifs et `False` sinon. Proposer des exemples et tester votre fonction.

♡

Exercice n°5 Créer une fonction `multiple7(L:`**list**`)->`**list** qui prend en argument une liste L et qui renvoie une liste qui contient les éléments de L multiple de 7 (dans le même ordre, avec éventuelllement des doublons). Par exemple : `multiple7([7, 3, 14, 7])-> [7, 14, 7]`

♡

```
multiple7([1, 2, 3])-> []
multiple7([0, 14, -7])-> [0, 14, -7]
```

Exercice n°6 Créer une fonction `suppr_doublon(L:`**list**`)->`**list** qui prend en argument une liste L et qui renvoie une liste qui contient les éléments de L, dans le même ordre, mais sans doublons. Par exemple :

```
multiple7([7, 3, 14, 7, 3, 14])-> [7, 3, 14]
multiple7([1, 2, 3])-> [1, 2, 3]
multiple7([0, 0,0])-> [0]
```

Exercice n°7 Écrire une fonction `concatene_all(L)` qui prend en argument une liste de liste et renvoie une liste formée de toutes les listes concaténées. Par exemple `concatene_all([ [1, 2, 3], [-1, -2], [4]])` renvoie `[1, 2, 3, -1, -2, 4]`.

II Exercices de mises en pratique

Exercice n°8 Écrire une fonction **zip** prenant en argument deux listes $L1$ et $L2$ et renvoyant la liste « zippé », c'est-à-dire constituée du premier élément de $L1$ plus le premier de $L2$ puis le deuxième de $L1$ etc...
Par exemple si `t1 = [31,28,31,30,31,30,31,31,30,31,30,31]` et `t2 = [ '␣Janvier␣', '␣Février␣', '␣Mars␣', '␣Avril␣', '␣Mai␣', 'Juin␣','␣Juillet␣', '␣Aout␣', 'Septembre␣', '␣Octobre␣', '␣Novembre␣', 'Décembre␣']`, alors **zip**`(t1, t2)` renverra `['Janvier',31,'Février',28,'Mars',31, etc...]`.

Exercice n°9 On s'intéresse au triangle de Pascal. Écrire une fonction `ligne_suivante(L.`**list**`)->`**list** qui prend en argument une liste L contenant les éléments d'une ligne du triangle de Pascal et qui en déduit (et renvoie) la ligne suivante. On se rappellera qu'un élément d'une ligne est obtenu en sommant l'élément immédiatement au-dessus et celui au-dessus à gauche, et les «trous» comptent comme des 0. Par exemple : `ligne_suivante([1,2,1])->[1,3,3,1]`

Exercice n°10 Écrire une fonction `Pascal(n:`**int**`)->`**list**`[`**list**`]` qui renvoie les n premières lignes du triangle de Pascal sous forme d'une liste de liste.

Exercice n°11 Créez une fonction prenant en argument un entier n et qui cherche tous les triplets d'entiers $(i,j,k) \in [\![1,n]\!]^3$ vérifiant la relation $i^2+j^2=k^2$. On essayera (si besoin dans un second temps) d'éviter d'afficher les doublons pour i et j dont le role est symétrique (ainsi, si 3,4,5 est solution, 4,3,5 l'est évidemment et il n'est pas nécessaire de le chercher/renvoyer).
Par exemple, `exo14(20)` pourra renvoyer `[(3, 4, 5), (5, 12, 13), (6, 8, 10), (8, 15, 17), (9, 12, 15)], (12, 16, 20)]`.

Exercice n°12 Écrire une fonction `premier(n)` qui renvoie `True` si le nombre n est un nombre premier et `False` sinon. On pourra utiliser le modulo pour tester si un nombre est divisible par un autre.

Exercice n°13 En utilisant la fonction précédente, écrire une fonction `premier_all(n)` prenant en argument un entier n et renvoyant la liste des nombre premiers inférieurs ou égaux à n.

Exercice n°14 Soit N un entier naturel. Ici, on testera avec $N=350$. Le but est d'implémenter le crible d'Erathostène afin d'obtenir une liste de nombres premiers de façon plus efficace que dans la question précédente. Le principe est le suivant :

- faire une liste de 351 objets True.
- Mettre les objets d'indice 0 et 1 à False
- On recherche le prochain indice i tel que $L[i]=True$, et on met tous les éléments de la liste dont l'indice est un multiple de i à False.
- On recommence l'opération. Ainsi, tous les entiers i tels que $L[i]=True$ sont premiers, et les autres non.
- À l'aide de la liste précédente, créer la liste des nombres premiers inférieurs ou égaux à 350.
- Faire une fonction renvoyant grâce à la méthode décrite ici la liste des nombres premiers de 1 à N (N étant l'argument de la fonction).
- Comparer la vitesse d'exécution de cette fonction avec celle de la question précédente. On utilisera pour cela le module time de la façon suivante : **import** `time` puis `deb = time.perf_counter()` avant d'appeler la fonction et enfin `fin = time.perf_counter()` après avoir appelé la fonction ; le temps écoulé est alors `delta_t = fin-deb` .

III À faire pour la prochaine séance

Exercice n°15 Écrire une fonction `multiple_de_3(L)` qui prend en argument une liste et qui renvoie la liste contenant les éléments de L qui sont multiples de 3. Par exemple `multiple_de_3([2, 4, 0, -3 , 4, 6])` doit renvoyer `[0, -3, 6]`. Dans le cas où aucun élément n'est multiple de 3, la fonction devra renvoyer la liste vide.

Exercice n°16 Écrire une fonction `somme_carre(n,m)` qui calcule $\sum_{i=1}^{n}\left(\sum_{j=1}^{m}(i+j)^2\right)$, on pourra utiliser deux boucles imbriquées et/ou une sous fonction. Attention à respecter le "point de départ et d'arrivée" des sommes (voir par exemple le paragraphe I).

Exercice n°17 Écrire une fonction `mini2(L)` qui prend en argument une liste et renvoie **l'indice** du minimum des éléments de cette liste.

Exercice n°18 On souhaite améliorer la fonction `mini2(L)` précédemment défini en spécifiant le comportement qu'elle doit avoir si le minimum apparait plusieurs fois dans la liste : `mini2a(L)` devra renvoyer l'indice du premier élément égal au minimum alors que `mini2b(L)` renverra l'indice du dernier élément égal au minimum. Par exemple
`mini2a([-1, 2, -1, 5])` renverra 0 alors que `mini2b([-1, 2, -1, 5])` renverra 2.