TP d'informatique n°1 : listes et boucles « for »

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 Avant le tp

Exercice n°1 Réviser la syntaxe d'une boucle **for** se répétant n fois

Exercice n°2 Réviser les manipulations de listes, par exemple :

- rappeler comment créer une liste contenant les éléments $3.4, 7, -10.2$;
- rappeler deux moyen d'accéder à l'élément 3.4 de la liste précédente ;
- rappeler comment concaténer deux listes ;
- rappeler comment ajouter un élément à la fin d'une liste (sans concaténation) ;
- rappeler comment accéder au nombre d'éléments d'une liste.

Vous pouvez éventuellement regarder les vidéos :
`https://youtu.be/dBr_kgtCGUM` `https://youtu.be/2fRrpyeF6BE`

II Boucles simples

Exercice n°3 Écrire une fonction `belle_fonction` prenant en argument un entier n et affichant n fois le message "la physique est la plus belle matière avec l'informatique".

Exercice n°4 Écrire une fonction `somme_entier` prenant en argument un entier n et renvoyant la somme $\sum_{k=0}^{n-1} k = 0 + 1 + 2 + \cdots + (n-1)$. (Il existe bien sûr une formule mathématique permettant de calculer cela, mais l'idée est de le faire avec une boucle) . On pourra tester : `somme_entier(10)= 45 ; somme_entier(15)= 105`.

Exercice n°5 Écrire une fonction `somme_cos(n:int)->float` prenant en argument un entier n et renvoyant la somme $\sum_{k=0}^{n-1} \cos(2\pi \times k/n)$. Testez votre fonction avec $n = 2$ et vérifiez le résultat à la main. Testez ensuite pour n quelconque.(On pourra importer la fonction cosinus et le nombre π de la bibliothèque math)

Exercice n°6 Écrire une fonction `factorielle(n:int)->int` prenant en argument un entier n et renvoyant la grandeur $n!$. Testez le pour 0, 5 et 10 ($0! = 1$ et $5! = 120$).

Exercice n°7 Écrire une fonction `cosinus(x:float, n:int)->float` prennant en argument un nombre x et un entier n (on donnera à n la valeur par défaut 10) et renvoyant la somme $\sum_{k=0}^{n-1}(-1)^k \times \frac{x^{(2k)}}{(2k)!}$
`cosinus(0.3)` : $0.95533648\ldots$; `cosinus(0.3,2)` : 0.955; `cosinus(0.3,1)` : 1.0;
`cosinus(0.9,15)` : $0.621609968\ldots$; `cosinus(3.14)` : $-0.999998735\ldots$;

Exercice n°8 On considère la suite définie par la relation de récurence $u_{n+1} = \sqrt{u_n + 2} + 0{,}5 \times u_n$ et son premier terme $u_0 \geqslant -2$. Sans utiliser les listes, écrire une fonction `calcule_u(u0:float,n:int)->float` qui prend en argument le premier terme de la suite en un entier n et qui renvoie la valeur de u_n correspondante.
On testera la fonction avec les valeurs suivantes : `calcule_u(1000,11)` : $8.689621\ldots$; `calcule_u(3,0)` : 3; `calcule_u(-1.5,1111)` : $5.464102\ldots$

Exercice n°9 La suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$. Écrire une fonction `fibo(n:int)->int` qui prend en argument un entier n et qui renvoie la valeur de F_n. Ici aussi on évitera d'utiliser les listes. On testera la fonction avec les valeurs suivantes : `fibo(10)` : 55; `fibo(16)` : 987; `fibo(1)` : 1; `fibo(0)` : 0;

III Listes et boucle for

Exercice n°10 Écrire une fonction `somme(L:list)->float` qui prend en argument une liste et renvoie la somme des éléments de la liste. Par exemple si `L = [1,3,6,10]` alors l'expression `somme(L)` devra renvoyer la valeur 20.

Exercice n°11 Écrire une fonction `moyenne(L:list)->float` qui prend en argument une liste et renvoie la moyenne des éléments de la liste. Par exemple si `L = [1,3,6,10]` alors `moyenne(L)` devra renvoyer la valeur 5.0.

Pour résoudre un exercice ou un problème c'est une bonne idée de penser à utiliser des fonctions que l'on a déjà codé pour éviter de ré-écrire plusieurs fois la même chose et pour rendre le code plus lisible.

Ici, je vous donne des exemples de résultats de la fonction. Ce n'est pas toujours le cas et c'est une bonne idée AVANT de commencer à coder de se poser cette question et de trouver quelques exemples simples qui permettront de tester la fonction ensuite.

Exercice n°12 Écrire une fonction `cherche(L:list, x:float)->bool` qui prend en argument une liste de nombre et un nombre et renvoie vrai si le nombre est dans la liste et faux sinon (`True` et `False` en python).

Exercice n°13 Écrire une fonction `maxi(L:list)->float` qui renvoie le plus grand nombre de la liste L. Par exemple si $L = [4, 7, 9.3, 5, 3]$ alors la fonction doit renvoyer 9.3.

Exercice n°14 Écrire une fonction `maxi2(L:list)->(int,float)` qui renvoie le plus grand nombre de la liste L et son indice sous forme d'une liste de deux éléments. Par exemple si $L = [4, 7, 9.3, 5, 3]$ alors la fonction doit renvoyer `[9.3, 2]`

Exercice n°15 Écrire une fonction `sous(tab1, tab2)` qui prend en argument deux listes de nombre de même longueur et qui renvoie la liste de même longueur contenant la soustraction terme à terme des deux listes (la première moins la deuxième). Par exemple `sous([1, 2, 2], [1, 1, 2])` doit renvoyer `[0, 1, 0]`.

Exercice n°16 Écrire une fonction qui prend en argument deux liste `nombre` et `coef` et renvoie la moyenne des nombres pondérés par les coefficients coef.
Par exemple si `nombre = [10, 10, 20]` et `coef = [1, 1, 2]` alors la moyenne pondérée est : $\frac{1\times10+1\times10+2\times20}{1+1+2} = 15$.

IV À faire pour la prochaine séance

Exercice n°17 Écrire une fonction `somme_carre_entier` qui prend en argument un entier n et renvoie la somme des carrés des entiers de 0 à n. Vous pouvez utiliser la formule analytique $n(n+1)(2n+1)/6$ pour vérifier que vous obtenez le bon résultat, mais il est interdit de l'utiliser dans votre programme.

Exercice n°18 Écrire une fonction `add_liste(L1,L2)` qui prend en argument deux listes (que l'on supposera de même longueur) et qui renvoie la liste contenant la somme terme à terme. Par exemple `add_liste([2, 4, 0], [-3, 4, 1])` doit renvoyer `[-1, 8, 1]`

Exercice n°19 Écrire une fonction `mini(L)` qui prend en argument une liste et renvoie le minimum des éléments de cette liste.

Exercice n°20 Écrire une fonction `mini_liste(L)` qui prend en argument une liste de liste et renvoie la liste des minimums de chacune des listes de cette liste.
Par exemple `mini_liste([[2, 4, 0], [-3, 4], [5, 7, 6, 10]])` doit renvoyer `[0, -3, 5]` (0 est le minimum de la première liste, -3 de la deuxième et 5 celui de la quatrième).