I_{03} Structure répétitive

PCSI 2020 – 2021

Un des gros avantages de l'ordinateur est sa capacité à répéter un très grand nombre de fois une même action. C'est ce que nous allons apprendre à faire ici.

I Boucle inconditionnelle

Si le nombre de répétitions (d'itérations) est connu à l'avance, on peut utiliser la structure for variable in range(n) :

```
for i in range(4):
    carre = i ** 2
    print(carre)
#fin de la boucle
print("fin")
```

⚠ La variable prendra les valeurs allant de 0 à n-1.

Comme pour les fonctions, l'intérieur de la boucle (qui sera répété plusieurs fois) se repère grâce aux indentations.
Lien Python tutor

1. Calculer une somme

On se propose de calculer $\sum_{k=0}^{9} \cos(2\pi k/10)$

La méthode classique est de définir une variable dans laquelle nous allons ajouter les termes un par un. Il ne faut surtout pas oublier d'initialiser la variable.

```
import math as m
s = 0 #initialisation
for k in range(10):# k : 0=>9
    s = s + m.cos(2*m.pi*k/10)
#fin de la boucle
```

Lien Python tutor

2. Calculer un produit

La méthode est la même, mais il ne faut pas oublier d'initialiser la variable à 1 et non à 0.
k démarre à 0, il faut être prudent pour éviter de se retrouver avec une somme nulle.

```
p = 1 #initialisation
for k in range(5):# k : 0=>4
    p = p*(k+1) # k+1 : 1=>5
```

Lien Python tutor

3. Création d'une liste

```
L = [0] * 5
for i in range(5):
    L[i] = i**2
```

```
L = []
for i in range(5):
    L.append(i**2)
```

Les deux méthodes fonctionnent. La méthode de gauche est préférable si on connait à l'avance le nombre de terme (comme ici), celle de droite si on ne connait pas à l'avance le nombre de terme.

4. Parcours de deux liste en même temps

Pour parcourir deux listes en même temps, on utilise les indices. Par exemple $\sum_{i=0}^{2} L1_i \times L2_i$

```
L1 = [1, 2, 3]
L2 = [4, 5, 6]
s = 0
for i in range(len(L1)):
    s = s + L1[i]*L2[i]
```

5. Parcours d'un tableau à 2D

Il est possible d'imbriquer une boucle for dans une autre.

```
M = [ [1,2,3], [4,5,6] ]
for i in range(len(M)): #len(M) = nombre de lignes
    for j in range(len(M[0])):# nombre de colonnes
        print(M[i][j])
```

On peut s'en servir pour parcourir une matrice (tableau 2D), calculer des tables de multiplications etc....
Lien Python tutor

6. Itérer sur une liste

Python permet un raccourci si on n'a pas besoin des indices et que l'on veut parcourir les éléments d'une liste.
Il est conseillé de prendre un nom de variable indiquant que l'on ne manipule PAS un indice val, x, mais pas i,j,k.

```
M = [0, -2, 4, 3, 0, 7]
for val in M:
    print(val)
```

7. Enumerate

Python permet un raccourci si l'on a besoin des indices et des valeurs d'une liste. Il est conseillé de prendre des noms de variables indiquant ce que l'on manipule ind,val pour indice et valeur.

```
M = [0, -2, 4, 3, 0, 7]
for ind, val in enumerate(M):
    print('ind =', ind)
    print('val =', val)
```

Lien Python tutor

⚠ Attention, il faut bien connaitre les deux premières syntaxes et ne pas les mélanger. Pour enumerate, je n'exige pas de vous que vous l'utilisiez mais je voudrais que vous soyez capable de le lire.

II Boucle conditionnelle

Si on ne connait pas le nombre d'itération à l'avance, il est possible d'utiliser la structure while.
Elle exécute un morceau de code tant qu'une condition est vraie.
Par exemple ci-dessous à droite, on recherche l'indice du premier terme strictement négatif dans une liste[1].

```
while condition: #s'arrête lorsque la condition est fausse
    calcul1
    calcul2
fin
```

```
def recherche_ind_neg(L):
    i = 0
    while L[i]>=0:#s'arrête lorsque L[i] < 0
        i = i+1
    return i
```

Pour ne pas vous tromper sur la condition à mettre dans le while, je vous conseille fortement d'écrire la négation de la condition en commentaire.

Il est très classique d'utiliser un compteur pour savoir combien d'itérations ont été faites.

```
compteur = 0
tant que xxx:
    faire yyy
    compteur += 1 #équivalent à compteur = compteur +1
```

1. Il aurait éventuellement fallu prendre des précautions si aucun terme n'est négatif.

III Exemples très importants

Ces exemples sont très importants, ils sont donc à comprendre, apprendre par cœur, savoir refaire, savoir adapter à des cas proches.

1. Somme des éléments d'une liste

Écrire une fonction prenant en argument une liste de nombre et renvoyant la somme des éléments de la liste.

```
def somme(liste):
    s = 0 #indispensable d'initialiser la somme
    for ind in range(len(liste)):
        #ind parcourt les indices 0...len(liste)-1
        s = s + liste[ind] #on ajoute l'élément dans notre somme partielle
    return s
```

```
def somme_bis(liste):
    s = 0 #indispensable d'initialiser la somme
    for x in liste:#x prend les différentes valeurs
        s = s + x #on ajoute l'élément dans notre somme partielle
    return s
```

Lien Python tutor

2. Détermination du minimum d'une liste d'élément

Écrire une fonction prenant en argument une liste de nombre et renvoyant le nombre le plus petit de la liste.

```
def mini(liste):
    mini_local = liste[0] #indispensable d'initialiser le mini_local, comme on ne sait pas quelle peut-être la plus petite valeur, on prend la première de la liste.
    for val in liste:#val prend les différentes valeurs
        if val < mini_local:#on test si notre mini_local est toujours le bon
            mini_local = val # si ce n'est pas le cas, on le change
        #sinon ... on ne fait rien ! (on garde notre minimum local)
    return mini_local
```

Lien Python tutor

3. Test d'appartenance d'un élément à une liste

Écrire une fonction prenant en argument un nombre et une liste de nombre et renvoyant vrai (`True`) si le nombre appartient à la liste et faux (`False`) sinon.

3.a. Une erreur classique

```
def test_faux(x,liste):
    for val in liste:
        if val == x:
            return True
        else:#val!=x
            return False
```

Exercice n°1 Expliquer pourquoi ce programme est faux. On pourra prendre un exemple.

3.b. Un programme juste

```
def test_juste(x,liste):
    for val in liste:
        if val == x:
            return True
        #else RIEN DU TOUT
    #je suis sorti du for sans jamais return, c'est donc que pour tout val dans liste, val != x, soit x n'appartient pas à liste
    return False #attention à l'indentation
```

3.c. Un programme juste avec un while

```
def test_avec_un_while(x,liste):
    trouve = False #au début, on ne sait pas si x est présent dans la liste
    ind = 0 #indice
    while not(trouve) and ind < len(liste):#tant que l'on n'a pas trouvé et qu'il reste des éléments dans la liste à tester
        #on s'arrête si on trouve, ou si il n'y a plus d'élément à tester
        trouve = x == liste[ind] #si x == liste[ind], trouve passe à vrai et on sort de la boucle
        ind = ind + 1
    return trouve
```

Lien Python tutor

Table des matières