Fiche 3 - Structures conditionnelles et répétitives

Il y a quatre structures différentes qui organisent un algorithme (ou programme) :

⇛ Le **branchement** : faire appel à un autre algorithme, par exemple l'appel à une fonction.

⇛ La **séquence** : suite d'instructions à exécuter l'une après l'autre, dans l'ordre d'écriture.

⇛ La **structure conditionnelle** ou l'**alternative** : effectue selon la valeur d'une condition booléenne (un test) une instruction ou éventuellement une autre.

⇛ Les **structures répétitives** ou **boucles** : répète une instruction un nombre de fois qui peut être déterminé à l'avance ou dépendre d'une condition booléenne.

Attention ! Certains langage acceptent une souplesse dans la rédaction d'un script. Par exemple, l'alignement des instructions (dans le cas d'une séquence d'instructions) n'est pas imposé ; ou encore, les structures conditionnelles ou répétitives possèdent des mots de début et de fin qui encadrent les instructions à exécuter :
`if ... then ... end` ou `for ... do ... end`

➜ Avec PYTHON, la rédaction est très contrainte :
➢ Les alignements doivent impérativement être respectés.

➢ Les instructions d'une structure conditionnelle ou répétitives sont décalées exactement de 1 ou de 4 espaces par rapport à l'alignement environnant.

Ceci est souvent une source d'erreur. Dans l'éditeur, un moyen efficace pour ne pas faire d'erreur est de valider chaque ligne avant d'écrire la suivante ; ce dernier gère de lui-même les contraintes de rédaction.

I. STRUCTURE CONDITIONNELLE : if

Il y a plusieurs variantes pour l'instruction conditionnelle : `si ... alors ...`

⇛ Cas simple :

```
if test:
    instructions pour le cas vrai;
```

⇛ Avec une alternative :

```
if test:
    instructions pour le cas vrai;
else:
    instructions pour le cas faux;
```

⇛ Avec plusieurs alternatives :

```
if test1:
    instructions pour le cas test1 vrai;
elif test2:
    instructions pour le cas test2 vrai;
elif test3:
    instructions pour le cas test3 vrai;
else:
    instructions pour les autres cas;
```

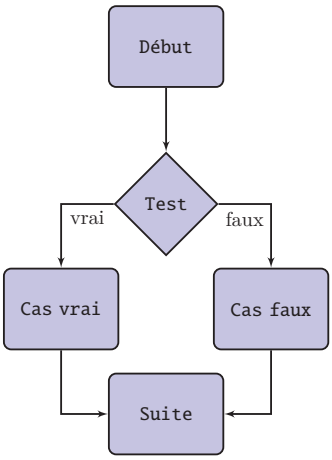
Remarque : Les tests sont exclusifs, dès que l'un est vérifié, les suivants ne sont plus considérés (même s'ils peuvent être vrais simultanément).

Exemple dans la shell

```
>>> a=float(input('Donner un nombre : a = '))
Donner un nombre : a = 1.234
>>> if a>2:
...  print('test1') # decalage : 1 espace
... elif a<2:
...  print('test2')
... elif a>1:
...  print('test3')
... else:
...  print('autre')
...
test2
```

Remarque : Les instructions `elif` et `else` sont optionnelles.

```
>>> if 12%2==0:
...     print('12 est pair') # decalage : 4 espaces
...
12 est pair
```



Lycée Saint-Exupery – maffre.saintexupery@gmail.com – MPSI

⇛ Exemple du calcul du maximum de deux nombres :

Script maximum.py

```
a=float(input("Premiere valeur a = "));
b=float(input("Seconde valeur b = "));
if a>b:
    M=a;
else:
    M=b;
print("Le maximum de ",a,' et ',b,' est ',M);
```

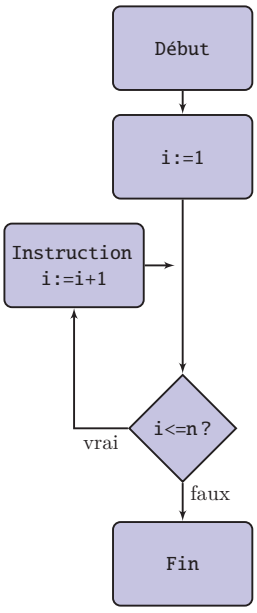
II. Structure répétitive for

Une boucle :
pour i variant de 1 à n faire instructions fin
se traduit en Python par

```
for i in range(1,n+1):
    instructions
```

➜ Situation d'utilisation : nombre d'itérations connu !

```
>>> for i in range(1,12,2):
...  print(i,end=' ')
...
1 3 5 7 9 11
>>> for i in range(1,12):
...  if i%2==1:
...   print(i,end=' ')
...
1 3 5 7 9 11
```



Remarque : L'ensemble de parcours de l'indice de la boucle est un objet itérable : liste, une chaîne de caractères, un tableau ligne ...

```
>>> a='St-Exp'
>>> for i in a:
...  print(i,end='_')
...
S_t_-_E_x_p_

>>> a=[1,2,3,4]
>>> for i in a:
...  print(i,end=' ')
...
1 2 3 4
```

⇛ Exemple d'une somme :

Voici un script que calcule $\sum_{k=1}^{n} \frac{1}{n^2}$

```
n=int(input('Donner n : '))
s=0 # initialisation
for k in range(1,n+1):
    s=s+1/k**2
print('Pour n =',n,'on a S =',s)
```

Attention ! Penser à rajouter le terme suivant par `s+=...` ou `s=s+...` et pas seulement faire une affectation qui effacerait la valeur de s sans tenir compte des calculs déjà effectués.

Remarque : On remarque l'initialisation de s à 0. Il est possible d'initialiser s au premier terme de la somme et d'adapter l'ensemble de variation de l'indice de boucle.

➜ Dans le cas d'un produit, il convient d'initialiser la variable à 1 ou à la valeur du premier facteur.

Remarque : Les instructions peuvent ne pas dépendre de la variable qui identifie l'étape de la boucle

⇛ Exemple d'une suite récurrence :

On souhaite calculer $u_n = \frac{x^n}{n!}$ en calculant par récurrence :

$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N} \quad u_n = \left(\frac{x}{n}\right) u_{n-1} \end{cases}$$

Script suite.py

```
print('Calcul de x^n/n!');
x=float(input("x = "));
n=int(input("n = "));
u=1;  # Initialisation : u_0
for i in range(1,n+1):
    u=u*x/i;  # Recurrence : calcul de u_i
print(str(x)+'^'+str(n)+'/'+str(n)+"! = ", u);
```

⇒ Exemple d'un système récurrent d'ordre 1.

Le programme suivant calcule le terme de rang n des suites (a_n), (b_n) et (c_n) définie par :

$$\begin{cases} a_{n+1} = a_n - nb_n \\ b_{n+1} = a_n c_n \\ c_{n+1} = a_n - (n-1)c_n \end{cases} \quad \text{avec } a_0 = b_0 = 1 \text{ et } c_0 = -2$$

Système récurrent

```
n=int(input("n = "));
a,b,c=1,1,-2
for i in range(1,n+1):
    a,b,c=a-(i-1)*b,a*c,a-(i-2)*c
print('a'+str(n)+' =',a);
print('b'+str(n)+' =',b);
print('c'+str(n)+' =',c);
```

Attention ! Une règle de lisibilité consiste à définir le terme de rang `i` lors de l'étape `i` de la boucle. Cependant, ici, le terme de rang `n+1` dépend de `n`. Ainsi, lors du calcul du terme de rang `i` il faut penser à réécrire la relation de récurrence :

$$\begin{cases} a_i = a_{i-1} - (i-1)b_{i-1} \\ b_i = a_{i-1}c_{i-1} \\ c_i = a_{i-1} - (i-2)c_{i-1} \end{cases}$$

en particulier, remplacer `n` par `i-1`.

Remarque : Lorsque la récurrence est d'ordre multiple, une méthode consiste à se ramener à un système récurrent d'ordre un.

Exemple, $u_{n+3} = u_n u_{n+2} + n u_{n+1}$ se réécrit :

$$\begin{cases} a_n = b_{n-1} \\ b_n = c_{n-1} \\ c_n = a_{n-1}c_{n-1} + (n-3)b_{n-1} \end{cases} \quad \text{avec} \quad \begin{cases} a_0 = u_0 \\ b_0 = u_1 \\ c_0 = u_2 \end{cases}$$

Alors $u_n = c_{n-2}$ (si $n \leq 2$).

III. Structure répétitive `while`

Une boucle :

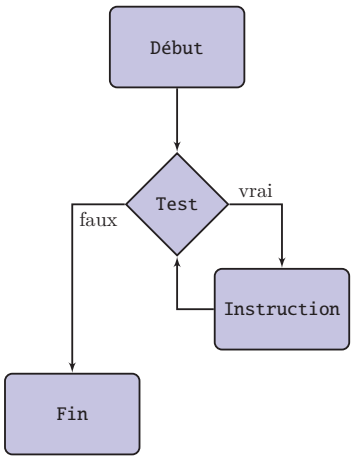
`tant que` condition `faire` instructions `fin`

se traduit en Python par

```
while condition:
    instructions
```

➜ Situation d'utilisation : le nombre d'itérations dépend d'une condition !

Attention ! Il convient par une étude préalable de vérifier que la condition de sortie de la boucle `while` se réalise en temps raisonnable ; sinon, la boucle peut se continuer indéfiniment.



⇒ Exemple d'un somme divergente :

La suite $\left(\sum_{k=1}^{n} \frac{1}{k}\right)_{n\in\mathbb{N}}$ est croissante et tend vers $+\infty$. Pour un nombre `A` donné, on veut détermine le plus petit n tel que

$$\sum_{k=1}^{n} \frac{1}{k} \geq A$$

```
A=float(input('Donner A : '))
u,n=1,1
while u<A:
    n,u=n+1,u+1/(n+1)
print(n)
```

Attention ! La condition d'une boucle `while` est la négation de ce que l'on recherche.

Remarque : Il est possible d'imbriquer des structures répétitives.

IV. Les mots clés `break` et `continue`

Les mots clefs `break` et `continue` permettent d'interrompre une boucle, ou de passer à l'itération suivante sans exécuter le reste du code.

Mot clef `break`

```
for i in range(10)
    if i==4:
        break
    print(i)
print('fin')
```

Ici, la boucle est interrompue lorsque $i = 4$. Python affiche donc successivement 0, 1, 2, 3 puis `'fin'`.

Mot clef `continue`

```
for i in range(10)
    if i%3:
        continue
    print(i)
print('fin')
```

Ici, on passe directement à l'itération suivante lorsque i n'est pas un multiple de 3. Python affiche donc successivement 0, 3, 6 puis `'fin'`