Numérique et sciences informatiques
Classe de première

A Langages et programmation
3. Structures itératives

I. Structure itérative bornée : la boucle « for »

I.1 Rappels sur les « boucles pour » en algorithmique

• Il est fréquent dans un algorithme de répéter plusieurs fois une même séquence d'instructions. Dans tout langage de programmation, on doit donc pouvoir demander à l'ordinateur d'exécuter un nombre donné de fois la même séquence d'instructions sans avoir à réécrire cette séquence d'instruction plusieurs fois.

• Par exemple, si on souhaite afficher les cinq premiers multiples de 7 strictement positifs, plutôt que d'écrire :

```
N <- 7

Afficher N
N <- N + 7

Afficher N
N <- N + 7

Afficher N
N <- N + 7

Afficher N
N <- N + 7
```

on préfère écrire :

```
N <- 7
Pour C de 1 à 5 :
        Afficher N
        N <- N+7
Fin Pour
```

• Ce type de structure s'appelle une **boucle** (*loop* en anglais) « pour ».

• Dans cet exemple, la séquence d'instruction sera exécutée cinq fois. Avant chaque nouvelle exécution de la séquence, la variable C, appelée **compteur de boucle** prendra une nouvelle valeur :
* avant la première exécution de la séquence, elle prend la valeur initiale (ici 1);
* avant chaque nouvelle exécution de la séquence, elle est augmentée (on dit **incrémentée**) d'un certain entier (égal à 1 lorsqu'on ne précise pas sa valeur). Cet entier est appelé le **pas d'incrémentation** ;
* à la dernière exécution, elle prend la valeur finale (ici 5).

• Le compteur de boucle peut être utilisé ou n'être pas utilisé dans le bloc d'instructions de la boucle. Dans notre premier exemple, il n'est pas utilisé mais on pourrait par exemple écrire :

```
N <- 0
Pour C de 1 à 5 :
```

```
        N <- N+7
        Afficher C
        Afficher N
Fin Pour
```

• Une boucle « pour » est aussi appelée une **boucle bornée** (ou une **structure itérative bornée**) car si on connaît :

* la valeur initiale du compteur de boucle ;
* sa valeur finale ;
* son pas d'incrémentation.

On peut donc connaître avant le début de l'exécution le nombre de fois que le bloc d'instructions sera répété.

-En algorithmique, il est très fortement déconseillé d'écrire dans le bloc d'instructions d'une boucle « pour » des instructions *qui modifient la valeur du compteur de boucle* car on risque alors de constater un comportement non prévu du programme.

I.2 Écrire une boucle « `for k in range(...)` » en python

On suppose dans cette partie que n, q et s représentent des entiers, donc des objets de type **`int`**.

a) Version 1 : on ne précise que la valeur finale du compteur

```
for k in range (n) :
    inst1
    ...
    instN
```

Remarques :

• Si $n \geq 1$, le bloc d'instructions sera exécuté n fois et le compteur de boucle `k` aura :

* **0** pour valeur initiale ;
* $\boldsymbol{n-1}$ pour valeur finale ;
* **1** pour pas d'incrémentation.

La variable `k` prendra donc les valeurs 0, puis 1, ... et enfin $n-1$ puis on « sortira » de la boucle `for`.

• Si la valeur de n est inférieure ou égale à 0, le bloc d'instructions n'est pas exécuté.

Les erreurs à éviter

• Ne pas oublier le signe **:**
• Ne pas oublier les parenthèses derrière de mot-clé **`range`**
• Ne pas oublier d'indenter le bloc d'instruction qui se trouve dans la boucle « `for` »
• Ne pas oublier que la dernière valeur prise par le compteur de boucle `k` sera `n - 1` et non `n`.

• Essayez vous-même : Dans IDLE, tapez l'instruction suivante :

```
>>>for k in range(5) :
        print(k)
```

Constatez que `k` est affiché 5 fois mais que la première valeur de `k` est 0 et la dernière est 4.

b) Version 2 : on précise la valeur initiale et la valeur finale du compteur

```
for k in range (q , n) :
  inst1
  ...
  instN
```

• Si q est strictement inférieur à n, le compteur de boucle k aura :
  * **q** pour valeur initiale ;
  * **n - 1** pour valeur finale ;
  * **1** pour pas d'incrémentation.

La variable `k` prendra donc successivement les valeurs `q`, `q + 1`, ... et enfin `n - 1`.

• Si `q` est supérieur ou égal à `n`, le bloc d'instructions n'est pas exécuté.

• Essayez vous-même : Dans IDLE, tapez l'instruction suivante :

```
>>>for k in range(15, 22) :
        print(k)
```

Constatez que la valeur de `k` est affiché 7 fois $(22 - 15)$ mais que la première valeur de `k` est 15 et la dernière est 21.

c) Version 3 : on précise la valeur initiale, la valeur finale du compteur et le pas d'incrémentation

```
for k in range (q , n, s) :
  inst1
  ...
  instN
```

• Si $s > 0$ et $q < n$ **ou** $s < 0$ et $q > n$, le compteur de boucle `k` aura :
  * `q` comme valeur initiale ;
  * `s` pour pas d'incrémentation ;
  * le « dernier » entier de la forme $q + l \times s$ « avant » `n` pour valeur finale.

La variable k prendra donc successivement les valeurs q, puis q + s, puis $q + 2s$, ... La dernière valeur de k sera celle obtenue *avant* d'atteindre la valeur n.

• Sinon, le bloc d'instructions n'est pas exécuté.

• Essayez vous-même : Dans IDLE, tapez l'instruction suivante :

```
>>>for k in range(11, 31, 5) :
        print(k)
```

Constatez que la première valeur de `k` est 11, que l'on incrémente de 5 et la dernière valeur est 26 et non 31.

• Tapez l'instruction suivante :

```
>>>for k in range(25, 5, -2) :
        print(k)
```

Constatez que la première valeur de k est 25, que l'on incrémente de -2 (c'est-à-dire que l'on décrémente de 2) et que la dernière valeur atteinte est 7 et non 5.

I.3 Écrire une boucle « for» sans utiliser la fonction `range(...)`

• En python, il est possible d'utiliser la boucle `for` pour « parcourir » certains objets dits **itérables**. Parmi les objets que nous avons rencontré jusqu'à présent, seuls les objets de type **`str`**, c'est à dire les chaines de caractères ont cette propriété.

• On peut ainsi obtenir successivement tous les caractères d'une variable de type str de la façon suivante :

```
for c in message :
    inst1
    ...
    instN
```

* ici, *message* est un objet de type str

* à chaque passage de boucle, la variable c prendra successivement pour valeur chacun des caractères qui forment la chaîne de caractère `message`.

• Essayez vous-même : Dans IDLE, tapez l'instruction suivante :

```
>>>for c in "Bonjour à tous !" :
        print(c)
```

Constatez que la première valeur de la variable `c` est `'B'` et la dernière valeur est `'!'`.

II Structure itérative non bornée : la boucle « `while` »

II.1 Rappels sur les boucles « tant que » en algorithmique

• La boucle « tant que » permet de répéter une séquence d'instructions *tant qu'*une certaine condition est vraie (et donc *jusqu'à ce qu*'elle devienne vraie). Par exemple, l'entier N prend initialement la valeur 1 et on veut le multiplier par deux jusqu'à ce qu'il prenne une valeur supérieure ou égal à 1000. On écrira alors le pseudo-code suivant :

```
N ← 1
Tant que N<1000 faire
    N ← N * 2
Fin Tant que
Afficher N
```

• La boucle « tant que » combine des propriétés de la structure conditionnelle « si ... » et de la boucle « pour ... » :

* comme la boucle « pour... » elle permet de répéter plusieurs fois une séquence d'instructions appelée **corps de la boucle** ;
* comme la structure « si ... », l'exécution de cette séquence d'instructions dépend du fait qu'une certaine expression booléenne renvoie la valeur « vrai » au moment où elle est évaluée.

• Une « boucle tant que ... » est aussi appelée une **structure itérative non bornée** (ou **boucle non bornée**) car on ne peut pas en général savoir à l'avance combien de fois la séquence d'instructions sera répétée puisque ceci dépend du nombre de fois où la condition associée à la boucle renverra la valeur « vrai ».

• Puisque c'est l'évaluation à « faux » qui conditionne le fait de sortir de la boucle « tant que », la condition sera toujours évaluée une fois de plus que le corps de la boucle sera exécutée.

Attention aux boucles infinies !

Lorsque l'on écrit une « boucle tant que », il faut veiller que la condition associée puisse devenir fausse à un moment donné de l'exécution du programme. En effet, si cette condition reste toujours vraie, le corps de la boucle sera exécuté un nombre infini de fois *et le programme ne s'arrêtera jamais*. On parle alors de **boucle infinie**.
Ce sont presque toujours les instructions contenues dans le corps de la boucle qui doivent faire évoluer la valeur renvoyée par la condition de la boucle pour qu'elle puisse passer de « vrai » à « faux ».

Utilisation possible d'un compteur de boucle

• Dans une boucle « tant que ... », la syntaxe ne prévoit aucune variable qui puisse servir de compteur de boucle permettant de savoir combien de fois le corps de la boucle a été exécuté à un moment donné du programme.

• En cas de besoin, le programmeur doit donc en créer un. Il faudra alors penser à l'incrémenter dans le corps de la boucle. Ainsi on peut modifier le code précédent de la façon suivante :

```
N ← 1
C ← 0
Tant que N < 1000 faire
    N ← N * 2
    C ← C + 1
```

Fin Tant que
Afficher N
Afficher C

Boucle « pour ...» et boucle « tant que ...»

Une boucle « pour » peut toujours être transformée en une boucle « tant que ». Par exemple la portion de programme suivante :

Pour C de 1 à 10 **faire**
    Afficher C
Fin Pour

peut être remplacée par la suivante qui produit exactement le même résultat :

$C \leftarrow 1$
Tant que C<=10 **faire**
    Afficher C
    $C \leftarrow C + 1$
Fin Tant que

• Toutefois, dans les cas où le nombre de répétition peut être connus à l'avance, il sera souvent préférable d'utiliser une « boucle « pour » plutôt qu'une boucle « tant que ».

II.2. Syntaxe de la boucle « tant que ...» en Python

En Python, on utiliser la syntaxe suivante pour créer une boucle « tant que » :

while *condition* **:**
    bloc1
bloc2

où :

* *condition* est une expression dont l'évaluation renvoie un objet de type **`bool`** (`True` ou `False`) ;
* *bloc1* est une séquence d'instructions exécutée à chaque passage de boucle, c'est-à-dire à chaque fois que l'évaluation de *condition* renvoie `True` ;
* bloc2 est une séquence d'instructions exécutée dès que l'évaluation de *condition* renvoie `False`.

• *S*i l'expression écrite à la place de *condition ne renvoie pas un objet de type* ***`bool`***, Python interprétera tout de même le résultat obtenu soit comme un **`True`** soit comme un **`False`**. Il est toutefois déconseillé pour un programmeur peu expérimenté d'exploiter cette propriété du langage.

• Comme pour la structure `if` ou la boucle `for` ,
* le signe **:** est obligatoire après la condition
* le bloc d'instructions qui constitue le corps de la boucle doit être *obligatoirement indenté*.

• Comme pour la structure `if`, si la condition renvoie **`False`** dès la première évaluation, le bloc d'instructions ne sera pas exécuté du tout.

• Essayez vous-même : Tapez dans IDLE, les instructions suivantes :

```
>>> N=1
>>> C=0
>>> while N < 1000 :
    N = N * 2
    C = C + 1
```

```
>>> print(N)
1024
>>> print(C)
10
```

Essayez d'autres valeurs dans la condition du `while`

III Les instructions `break` et `continue`

III.1 L'instruction `break`

• Il arrive que l'on souhaite « sortir » d'une boucle `while` avant que la condition associée soit devenue fausse ou « sortir » d'une boucle `for` avant que le compteur de boucle n'ait parcouru toutes les valeurs prévues. Dans ce cas on utilise le mot-clé **`break`**.

• Le mot clé break est utilisé sur une seule ligne et *constitue une instruction*. Elle consiste à « sortir » de la boucle dans laquelle on se trouve. La première instruction exécutée ensuite sera la première instruction qui se trouve après la boucle.

• Essayez vous-même : Tapez les instructions suivantes puis exécutez le programme :

```
>>> while True :
        rep=input("On continue ? 'n' pour non : ")
        if rep=='n' :
            break
```

Pas d'instruction break en dehors d'une boucle

Si on cherche à utiliser l'instruction **`break`** en dehors d'une boucle `for` ou d'une boucle `while`, un message d'erreur sera généré.

• Essayez vous-même : Tapez les instructions suivantes :

```
>>> break
SyntaxError: 'break' outside loop
```

III.2 L'instruction `continue`

• On peut utiliser l'instruction `continue` pour interrompre l'exécution de la séquence d'instructions dans le corps d'une boucle `for` ou `while` sans pour autant « sortir » de la boucle.

• L'exécution de l'instruction `continue` conduit à :

* aller directement au moment de l'évaluation de la condition si on se trouve dans une boucle la boucle `while` ;

* donner la prochaine valeur du compteur de boucle et reprendre au début du corps de la boucle dans une boucle `for`.

• Essayez vous-même : Tapez et exécutez le code suivant :

```
for c in range(4, 15, 2) :
    if c == 8 :
        continue
    print("c = ", c)
```

• Essayez vous-même : Tapez et exécutez le code suivant :

```
n = 5
while n > 0 :
    n = n - 1
    if n == 2 :
        continue
```

```
    print("n = ", n)
```