Chapitre 10 : les boucles imbriquées

Le but de ce chapitre est d'apprendre à analyser un problème contenant des répétitions imbriquées et à utiliser des boucles imbriquées pour traiter ces configurations.

Petit complément sur les boucles simples

D'un point de vue algorithmique on peut distinguer deux types de boucles simples :

- les **boucles incrémentées à pas constant** qu'on peut aussi appeler **boucles avec itérateur numérique**.

Ce type de boucle est utilisé lorsque le **nombre d'itération(s) est connu (ou calculé)** à l'avance, c'est-à-dire avant l'entrée dans la boucle.

Elle s'écrit de la manière suivante :

```
Instructions_ini # initialisation d'un itérateur numérique entier
while (condition): #
  Instruction 1
  Instruction 2
  ...
  Instruction n
  Instructions_Incr # incrémentation de l'itérateur
Instruction n+1
```

- La condition d'arrêt porte généralement sur la variable entière correspondant à l'itérateur. On teste souvent si la variable est inférieure ou supérieure à une borne (la limite).
- L'itérateur doit être initialisé avant l'entrée dans la boucle.

L'incrémentation de l'itérateur peut prendre plusieurs formes. Il est possible d'opérer des décréments plutôt que des incréments. Il est aussi possible d'incrémenter par des pas différents de `1` .

Exemple :

In [ ]:

```
it = 1
borne=11
while it<borne :
    print(it)
    it+=2


it=9
borne=0
while it>borne :
    print(it)
    it-=2
```

Ce type de boucle est par exemple utile pour parcourir un tableau.

- les autres boucles parfois appelés *boucles conditionnelles* ont une condition d'arrêt booléenne plus générale et le **nombre d'itérations n'est pas connu à l'avance (il peut être de 0)**. Par exemple :

In [1]:

```
print('entrez oui ou non')
choix=input()
while choix!='oui' and choix!='non' :
    print('entrez oui ou non')
    choix=input()
print('Vous avez choisi ', choix)
```

```
entrez oui ou non
h
entrez oui ou non
oui
Vous avez choisi  oui
```

Toute boucle qu'elle soit avec itérateur numérique ou conditionnelle, peut être codée au moyen d'un `while`. C'est ce qui sera fait dans ce cours. Certains langages proposent une syntaxe pour les boucles avec itérateur numérique : ce sont les boucles `for` qu'on trouve par exemple avec le langage C.

Les boucles imbriquées

- On dit que deux boucles sont *imbriquées* si **l'une est contenue dans le bloc d'instructions de l'autre**.
- Certains algorithmes nécessitent d'avoir une imbrication de boucles.

> C'est le cas si l'on souhaite par exemple afficher toutes les tables de multiplications de `1` à `10`. En effet,
>
> - l'affichage de la table de multiplication d'un nombre, par exemple la table de multiplication de `7`, nécessite une boucle ;
> - de plus, il faut afficher la table de multiplication pour chaque nombre entre `1` et `10`, ce qui nécessite une deuxième boucle contenant la première.

Premier exemple

Il s'agit d'afficher les tables de multiplication pour tous les nombres entre `1` et `10` inclus. Chaque table se présentera comme suit :

```
 TABLE de 4
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
```

Comme très souvent, plutôt que d'essayer d'écrire directement le détail du programme, il peut être préférable de procéder par étape. Par exemple, nous pouvons dire que, globalement, notre programme doit écrire les `10` tables de multiplication de `1` à `10` et qu'il doit donc se présenter ainsi :

```
i=1
while i<=10 :
            # écrire la table de i
   i+=1
```

Pour écrire la table du nombre `i` , nous pouvons procéder ainsi :

```
print("\n\nTABLE de ", i, '\n')
j=1
while j<=10 : #Pour chaque nombre j variant de 1 à 10
    print(i, " * ", j, " = ", i*j)
    j+=1
```

En insérant dans la boucle `while` principale le code permettant l'écriture de la table de multiplication, nous obtenons le code complet suivant :

In [ ]:

```
i=1
while  i<=10 :
    print("\n\nTABLE de ", i, '\n')
    j=1
    while j<=10 : #Pour chaque nombre j variant de 1 à 10
        print(i, " * ", j, " = ", i*j)
        j+=1
    i+=1
```

Remarque : On peut remplacer l'écriture de boucles imbriquées par l'appel de fonction. Par exemple, si l'on définit la fonction `tableMult` permettant d'écrire la table de multiplication d'un nombre `i` , l'algorithme devient :

In [ ]:

```
def tableMult(i) :
    """ Affiche la table de multiplication de i """
    j=1
    while j<=10 : #Pour chaque nombre j variant de 1 à 10
        print(i, " * ", j, " = ", i*j)
        j+=1

# algo principal
i=1
while  i<=10 :
    print("\n\nTable de ", i, '\n')
    tableMult(i)
    i+=1
```

La fonction `tableMult` contenant une boucle, il y a bien imbrication de boucles pour l'ordinateur mais le fait de transférer la boucle correspondant à l'écriture de la table de multiplication de l'entier `i` dans la fonction `tableMult` permet, lors de l'écriture de l'algorithme, de n'écrire qu'une simple boucle, **le programme est plus lisible**.

- Sur ce thème : **Exercice 1 et 3 du TD**

Deuxième exemple

On souhaite écrire un programme permettant de jouer au nombre magique. Voici le déroulement d'une partie :

1. l'ordinateur choisit un nombre aléatoire entre `1` et `100` inclus ;
2. le joueur a `10` tentatives pour déterminer ce nombre. À chaque tentative, si le nombre saisi par le joueur n'est pas le nombre aléatoire choisi par l'ordinateur, le programme indique si le nombre proposé par le joueur est plus petit ou plus grand que le nombre aléatoire.

On souhaite que le programme permette de jouer plusieurs fois de suite à ce jeu. Ainsi, après chaque partie, l'ordinateur demande à l'utilisateur s'il veut rejouer. Le programme s'arrête dès que le joueur ne veut plus jouer.

Pour réaliser un tel programme, il faut d'abord déterminer le code permettant de jouer au nombre magique. On a donc un nombre choisi aléatoirement et une boucle qui demande à l'utilisateur de saisir un nombre puis indiquant si celui-ci est plus petit ou plus grand que le nombre magique, et ce jusqu'à ce que l'utilisateur ait effectué `10` tentatives ou trouvé le nombre magique.

Un exemple de code est le suivant.

In [ ]:

```
from random import *
nombreAleatoire = randint(1,100)
tentative = 0
nombre = -1

while nombre !=  nombreAleatoire and tentative < 10 :
    tentative=+1
    print("Saisissez un nombre : ")
    nombre=int(input())
    if nombre < nombreAleatoire :
        print("Le nombre magique est plus grand\n")
    elif nombre > nombreAleatoire :
        print("Le nombre magique est plus petit\n")

if nombre==nombreAleatoire :
  print("Vous avez gagné\n")
else :
  print("Vous avez perdu\n")
```

Ce code doit être exécuté une première fois. Le programme doit ensuite demander à l'utilisateur s'il souhaite rejouer une autre fois et ceci doit se répéter tant que l'utilisateur saisit la lettre `O` pour oui. Cette boucle est alors de la forme :

In [ ]:

```
rejoue="O"
while rejoue=="O" :
    print("Nouvelle partie\n")

    #Code de la partie...

    print("Voulez-vous rejouer (O/N) ?")
    rejoue=input()
```

En insérant le code permettant de jouer à une partie, on obtient alors le programme complet suivant :

In [ ]:

```
from random import *

rejoue="O"
while rejoue=="O" :

    print("Nouvelle partie\n")

    nombreAleatoire = randint(1,100)
    tentative = 0
    nombre = -1
    while nombre !=  nombreAleatoire and tentative < 10 :
        tentative=+1
        print("Saisissez un nombre : ")
        nombre=int(input())
        if nombre < nombreAleatoire :
            print("Le nombre magique est plus grand\n")
        elif nombre > nombreAleatoire :
            print("Le nombre magique est plus petit\n")

    if nombre==nombreAleatoire :
        print("Vous avez gagné\n")
    else :
        print("Vous avez perdu\n")

    print("Voulez-vous rejouer (O/N) ?")
    rejoue=input()
```

Attention : Il faut faire très attention à l'initialisation de la boucle à l'intérieur. Pour chaque partie, il faut choisir un nombre aléatoire et mettre le compteur de tentatives à 0. Autrement dit, les instructions correspondantes (lignes 8 à 10) sont faites à l'intérieur de la boucle `while` et non au début du programme !

Comme dans l'exemple précédent, il est possible d'encapsuler la boucle d'une *partie du jeu nombre magique* dans une fonction `jouer_partie_nombre_magique` :

In [ ]:

```
def jouer_partie_nombre_magique() :
    """ Permet de jouer une partie du jeu nombre magique"""
    nombreAleatoire = randint(1,100)
    tentative = 0
    nombre = -1
    while nombre !=  nombreAleatoire and tentative < 10 :
        tentative=+1
        print("Saisissez un nombre : ")
        nombre=int(input())
        if nombre < nombreAleatoire :
            print("Le nombre magique est plus grand\n")
        elif nombre > nombreAleatoire :
            print("Le nombre magique est plus petit\n")

    if nombre==nombreAleatoire :
        print("Vous avez gagné\n")
    else :
        print("Vous avez perdu\n")
```

L'algorithme final devient :

In [ ]:

```
from random import *

rejoue="O"
while rejoue=="O" :

    print("Nouvelle partie\n")
    jouer_partie_nombre_magique()
    print("Voulez-vous rejouer (O/N) ?")
    rejoue=input()
```

- Sur ce thème : **Exercice 2, 4 et 5 du TD**