

CHAPITRE

4

DES BOUCLES POUR RÉPÉTER



Table des matières

1	La boucle <i>for</i>	2
2	La boucle <i>while</i>	5
3	Le coin des exercices :	6
4	Construction de patterns en console :	8
5	Retour vers Tortue	9

En programmation, on est souvent amené à répéter une ou plusieurs instructions : ce sont les *boucles* qui permettent de réaliser cette action. On se propose ici de les étudier particulièrement.

1 La boucle *for*

Certainement la plus utilisée, elle permet de *numéroter* le nombre de répétitions. Un exemple :

```
1 for i in range(10):
2     print("J'ai compris les boucles")
```

La boucle permet de répéter 10 fois l'instruction `print("J'ai compris les boucles")`.



La variable `i` est appelée *variable de compteur*.

À chaque fois que la variable entière `i` prend une valeur entre 0 et 9, le bloc d'instruction est exécuté : il y a donc 10 boucles.



Le nom de `i` est habituel : mais vous pouvez lui donner un autre nom...

Les instructions du bloc peuvent évoluer selon les valeurs de la variable `i` comme le montre l'exemple suivant :

```
1 for k in range(10):
2     print(2*k)
```

L'*indentation* est importante : elle permet d'indiquer quelles instructions sont répétées.

Exercice n°1

Quelle est la différence entre les deux codes suivants pourtant voisins ?

```
1 #---- code1.py ----#
2 u = 1
3 for i in range(5):
4     u = 2 * u
5     print(u)
6 #---- code2.py ----#
7 u = 1
8 for i in range(5):
9     u = 2 * u
10 print(u)
```

Exercice n°2

Donner la valeur de la variable `u` à la fin de la boucle.

Si l'indentation n'est pas respectée, un message d'erreur apparaît dans la console :

FIGURE 1 – Les messages d'erreurs sont en général implicites.

```
1 for i in range(10):
2 print(2*i)
```

Shell ×

Python 3.7.2 (bundled)

```
>>> %Run boucle2.py
Traceback (most recent call last):
  File "C:\Users\marc\AppData\Local\Programs\Thonny\lib\as
    return compile(source, filename, mode, PyCF_ONLY_AST)
  File "C:\Users\marc\Documents\Marc\NSI\Cours_NSI\Boucles
    print(2*i)
    ^
IndentationError: expected an indented block
```



Par défaut, `range(N)` itère de l'entier 0 à l'entier $N - 1$ (il y a bien N entiers) avec un pas de 1. Ces paramètres peuvent être redéfinis.

Le "print" est utilisé ici pour visualiser l'effet des instructions : il peut être utile pour visualiser l'évolution d'une variable dans un programme...

De façon générale, la boucle `for` est souvent utilisée pour parcourir tout *objet itérable* comme une chaîne de caractères ou une liste.

Exercice n°3

Recopier et exécuter le code suivant pour visualiser les valeurs prises par la variable `elt`

```
1 phrase = "J'aime les papates crues"
2 for elt in phrase:
3     print(elt, end = ',')
```

Exercice n°4

On considère le programme ci-dessous :

- ① Dans un tableau, écrire comment évoluent les variables i et S lors de l'exécution de ce code.
- ② À quoi sert-il ?

```
1 S = 0
2 for i in range(10):
3     S = S + i
4 print(S)
```

Retour sur le module Turtle. On a pu se rendre compte que la construction d'un triangle équilatéral, d'un carré nécessitait **la répétition** de deux instructions élémentaires :

- une avance de p pixels ;
- une rotation d'un angle α .

Cet angle α dépend du nombre de côtés n du polygone régulier :

n	3	4	5	nbre
α	120°	90°	72°	360/nbre
Nom	Triangle	Carré	Pentagone	Dépend de nbre

Il faut répéter les deux instructions autant de fois que la valeur du nombre de côté l'indique :

```
1 from turtle import *
2 dim = 100
3 nbre = int(input("Donner le nombre de côtés du polygone: "))
4 for i in range(nbre):
5     fd(dim)
6     lt(360/nbre)
```

2 La boucle *while*

Elle permet de répéter un bloc d'instructions **tant que** une condition est vraie. Elle nécessite pour être utilisée correctement :

- la condition doit être **initialisée** avant la boucle ;
- il doit y avoir **un test** dans la condition ;
- l'**état de la condition** doit être modifié dans la boucle.

L'exemple suivant permet de déterminer combien de fois il faut multiplier 1 par 2 pour dépasser 100 :

```
1  compteur = 0           # variable de type int initialisée à 0
2  u = 1                 # variable de type int initialisée à 1
3  while u < 100:       # la condition u < 100 peut être évaluée car la valeur de u
   ↪ est connue
4      u = 2*u
5      compteur = compteur + 1
6  print(compteur)
```

À chaque tour de boucles, l'expression `u < 100` est évaluée :

- ➔ si elle est **vraie**, alors les instructions du bloc sont exécutées ;
- ➔ si elle est fausse, on **sort** de la boucle/



Si la condition `u < 100` ne change pas d'état, vous risquez de provoquer une boucle **infinie** !

L'instruction `u = 2*u` permet de changer la valeur de la variable `u` : l'instruction `u < 100` est **vraie** au départ puis devient **fausse** au bout d'un moment : on sort alors de la boucle.

Exercice n°5

Que dire de tels codes ?

```
1  # ---- code1 ---- #           1  # ---- code2 ---- #
2  i = 0                           2  i = 10
3  while i < 4:                     3  while i < 5:
4      fd(50)                       4      print(i)
5      lt(90)                       5      i = i + 1
```

Néanmoins dans certaines situations, on pourra utiliser une boucle infinie ; par exemple un serveur qui écoute sur un port et attend qu'un client se manifeste.

On utilise alors une condition qui est toujours vraie comme **True**, variable booléenne (au tout autre valeur non nulle). L'instruction **break** permettra alors d'arrêter la boucle dès que survient un événement défini par l'utilisateur.

```

1 from random import randint
2 somme = 0
3 while True:
4     n = randint(0,100)
5     if n == 50:
6         break
7     somme = somme + n
8 print(f"Vous avez gagné {somme} euros")

```



Le code précédent utilise une *f-string* : c'est une chaîne de caractères dans laquelle est introduite la *valeur* d'une variable.

Nous allons en utiliser pour faire des affichages dynamiques.

3 Le coin des exercices :

Exercice n°6

Le code suivant utilise une boucle (pour répéter...) et une *f-string* (pour un affichage dynamique) qui permet d'afficher les 20 premiers carrés.

- ❶ Recopier le code ci-contre et exécutez le.
- ❷ Modifiez le code pour afficher les 100 premiers carrés.
- ❸ Modifiez le code pour afficher les 100 premiers cubes.

```

1 for i in range(20):
2     carre = i**2
3     print(f"Le carré de {i} est {carre}")

```

Exercice n°7

On cherche un code Python qui permet l'affichage des tables de multiplication. Recopier et compléter le code suivant qui permet d'afficher la table de multiplication de *n* ou *n* est un entier saisi au clavier...

```

1 n = int(input("Table de quel chiffre? "))
2 print(f"La table de {...} est:")
3 for i in range(10):
4     print(f"Le produit de {...} par {...} est {...}.")

```

Exercice n°8

Dans le tableau ci-contre, donnez l'évolution de la valeur des variables des instructions ci-après.

```

1 u = 2
2 s = 0
3 for i in range(5):
4     s = s + u
5     u = u + 4

```

<i>u</i>						
<i>s</i>						
<i>i</i>						

Exercice n°9

Voici un exemple de code contenant deux boucles.

- ❶ Combien d'affichages seront réalisés avec ces boucles ?
- ❷ Quel est le premier affichage du code précédent ?
- ❸ Quel est le 82ième affichage du code précédent ?

```

1 for i in range(10):
2     for j in range(10):
3         print(f"La somme de {i} et de {j} est {i + j}")
4     print("-----")

```

Exercice n°10

Qu'affiche ce programme lorsque l'utilisateur saisit :

- ❶ N = 51
- ❷ N = 456
- ❸ N = 1

```

1 N = int(input("Donne moi un nombre entier! "))
2 sol = 50
3 while sol < N:
4     sol = sol + 10
5 print(sol)

```

Exercice n°11

- ❶ Faites tourner cet algorithme à la main sur des entrées simples comme 78 ou 231.
- ❷ À quoi sert-il ?

Données : un nombre entier N et une liste vide L

Résultat : la liste L complétée

initialisation : effectuer la division euclidienne de N par 2 en trouvant le quotient Q et le reste R ;

ajouter R au début de L

tant que Q est différent de 0 **faire**

N prend la valeur Q

 Recommencer la division euclidienne de N par 2 en trouvant le quotient Q et le reste R

 ajouter R au début de L

fin

retourner L

Algorithme 1 : Un algorithme presque simple...

Exercice n°12

Voici le code d'un jeu mathématique : testez-le !

```
1 from random import randrange
2 print("Bienvenue au jeu des additions!")
3 score = 0
4 reponse = True
5 while reponse == True:
6     a = randrange(1,10)
7     b = randrange(1,10)
8     s = int(input(f"{a}+{b} donne combien? ... "))
9     if s == a + b:
10        score = score+1000
11        print(f"Correct! Votre score actuel est: {score} ")
12    else:
13        reponse = False
14    print(f"Faux! Votre score final est:{score} ")
```

4 Construction de patterns en console :

Exercice n°13

- 1 Recopier et exécutez le code suivant sur plusieurs valeurs.
- 2 À quoi sert l'instruction du deuxième `print()` ?



La méthode `print` par défaut, affiche dans la console, l'information passée en paramètre et effectue un retour à la ligne. En changeant l'attribut `end`, vous pouvez choisir ce que fait la méthode `print` après l'affichage. Dans le code suivant, l'instruction `print('*', end = " ")` affiche une étoile puis un espace, sans retour à la ligne !

```

1 num_ligne = int(input("Entrer le nombre de ligne: "))
2 for i in range(0, num_ligne):
3     for j in range(0, i + 1):
4         print('*', end = " ")
5     print()

```

Exercice n°14

Modifier légèrement le code précédent afin d'obtenir les motifs suivants.

<pre> Entrer le nombre de ligne: 5 * * * * * * * * * * * * * * * </pre>	<pre> Entrer le nombre de ligne: 5 * * * * * * * * * * * * * * * </pre>
---	---

Exercice n°15

Changer vos programmes en fonctions qui prendront en entrée le nombre de lignes et le caractère souhaité pour la construction des pyramides(ici une étoile...).

En s'appuyant sur les propriétés des codes précédents, réalisez les pyramides de nombres suivantes :

<pre> Donner le nombre de ligne: 5 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 </pre>	<pre> Donner le nombre de ligne: 5 1 1 2 1 2 3 1 2 3 4 1 2 3 4 5 </pre>	<pre> Donner le nombre de ligne: 6 1 2 3 3 4 5 4 5 6 7 5 6 7 8 9 6 7 8 9 10 11 </pre>
---	---	---

5 Retour vers Tortue

Exercice n°16

Tape ces instructions dans Thonny et exécute le code

```

1 from turtle import *
2 screen = Screen()
3 screen.setup(900,900)
4 screen.title('Jeu de dames')
5 speed(0)          #vitesse à fond

6 def carre(cote, c):
7     fillcolor(c)
8     begin_fill()
9     for i in range(4):
10        fd(cote)
11        lt(90)
12    end_fill()

13 exitonclick()

```



Tu as créé un nouvel outil *sans* l'utiliser : donc à l'exécution de ce code, il se passe apparemment rien...

Tu as en effet défini une fonction `carre` qui possède *deux paramètres* : à l'appel de la fonction, il faudra choisir la valeur de ces paramètres comme le montre les exemples suivants :

```
1 carre(60, 'red')
2 carre(100, 'green')
```

Exercice n°17

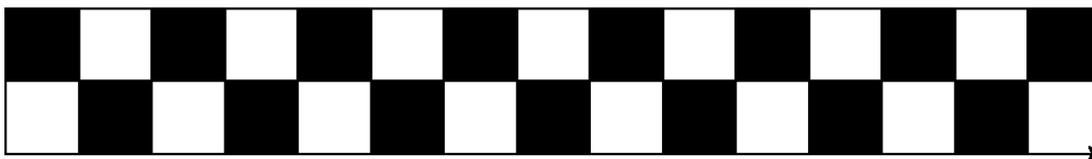
Donne les instructions qui permettent la construction de cette ligne. L'écran fait 900 pixels sur 900 pixels et chaque carré mesure 60 pixels de côté...

Par exemple,



Exercice n°18

Même construction en doublant la ligne et en alternant les couleurs



Et maintenant le damier complet !

Exercice n°19

Réalise le damier complet.



Utilise une fonction !

