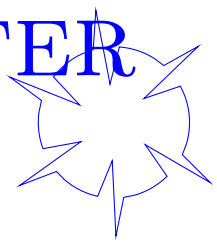
CHAPITRE

# 3 Des boucles pour répéter



## Table des matières

En programmation, on est souvent amener 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 Retour vers Turtle

Dans un travail sur la librairie `Turtle`, tu as appris à commander un stylo pour construire des figures géométriques. Très vite, est venu le besoin de ***répéter*** des instructions.
Par exemple la construction d'un carré nécessite la répétition de ***quatre*** blocs d'instructions. Nous utilisons donc une boucle pour cela :

```
for i in range(4):
    fd(100)
    lt(90)
```

On peut aussi intégrer la valeur de `i` dans le bloc d'instruction : c'est cette manipulation qui nous a permis de construire une spirale avec des triangles de plus en plus grand.

```
nbre = int(input("Nombre de motifs? "))

# --- Definition de la fonction triangle ---- #
def triangle(cote):
    for i in range(3):
        fd(cote)
        lt(120)

for i in range(nbre):
    triangle(50*(i + 1)) # appel de la fonction triangle
    lt(20)
```

À chaque tour de boucle, la dimension du triangle équilatéral change.

# 2 La boucle *for*

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

```
for i in range(10):
    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 choix de `i` est historique et vous pouvez l'appeler autrement...

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

```
for k in range(10):
    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 ?

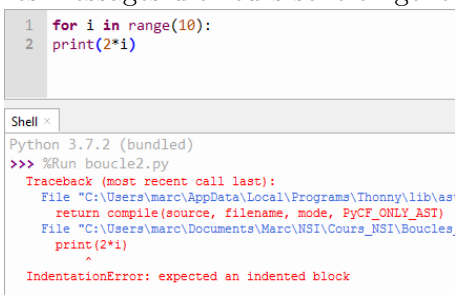
```
#----   code1.py     ----#
u = 1
for i in range(5):
    u = 2 * u
    print(u)
```

```
#----   code2.py     ----#
u = 1
for i in range(5):
    u = 2 * u
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.




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`

```
phrase = "J'aime les papates crues"
for elt in phrase:
    #elt est une variable qui prend successivement la valeur des caractères de la
    ↪ variable phrase
    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 ?

```
S = 0
for i in range(10):
    S = S + i
print(S)
```

**Exercice n°5**

Le triangle équilatéral (3 côtés) et le carré (4 côtés) font partie de la famille des polygones réguliers.

❶ Le polygone régulier à 5 côtés s'appelle un ***pentagone***. Écrire un programme Python qui permet de le construire.

❷ Compléter le tableau suivant :

| $n$ | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| `alpha` | 120° | 90° | ... | ... |
| Nom | Triangle | Carré | Pentagone | Hexagone |

❸ Compléter ensuite le programme Python suivant qui permet de choisir la construction d'un polygone régulier.

```
from turtle import *
speed(0)
dim = 100
nbre = int(input("Donner le nombre de côtés du polygone: "))
for i in range(nbre):
    fd(dim)
    lt(......)
```

# 3 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 :

```
compteur = 0          # variable de type int initialisée à 0
u = 1                 # variable de type int initialisée à 1
while u < 100:        # la condition u < 100 peut être évaluée car la valeur de u est
↪   connue
    u = 2*u
    compteur = compteur + 1
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°6**
Que dire de tels codes ?

```
#  ----   code1   ----  #
i = 0
while i < 4:
    fd(50)
    lt(90)
```

```
#  ----   code2   ----  #
i = 10
while i < 5:
    print(i)
    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.

```
from random import randint
somme = 0
while True:
    n = randint(0,100)
    if n == 50:
        break
    somme = somme + n
print(f"Vous avez gagne {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.

# 4 Le coin des exercices :

**Exercice n°7**

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.

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

**Exercice n°8**

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...

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

**Exercice n°9**

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

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

| | | | | | | |
|---|---|---|---|---|---|---|
| $u$ | | | | | | |
| $s$ | | | | | | |
| $i$ | | | | | | |

**Exercice n°10**

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 ?

```python
for i in range(10):
    for j in range(10):
        print(f"La somme de {i} et de {j} est {i + j}")
    print("-------------------------------------------")
```

**Exercice n°11**

Qu'affiche ce programme lorsque l'utilisateur saisit :

❶ N = 51

❷ N = 456

❸ N = 1

```python
N = int(input("Donne moi un nombre entier! "))
sol = 50
while sol < N:
    sol = sol + 10
print(sol)
```

**Exercice n°12**

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°13**

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

```python
from random import randrange
print("Bienvenue au jeu des additions!")
score = 0
reponse = True
while reponse == True:
    a = randrange(1,10)
    b = randrange(1,10)
    s = int(input(f"{a}+{b} donne combien? ... "))
    if s == a + b:
        score = score+1000
        print(f"Correct! Votre score actuel est: {score} ")
```

```
    else:
        reponse = False
        print(f"Faux! Votre score final est:{score} ")
```

# 5 Construction de patterns en console :

**Exercice n°14**

❶ Recopier et exécutez le code suivant sur plusieurs valeurs.

❷ 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 !

```
num_ligne = int(input("Entrer le nombre de ligne: "))

for i in range(0, num_ligne):
    for j in range(0, i + 1):
        print('*', end = " ")
    print()
```

**Exercice n°15**

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

```
Entrer le nombre de ligne: 5
* * * * *
* * * *
* * *
* *
*
```

```
Entrer le nombre de ligne: 5
        *
      * *
    * * *
  * * * *
* * * * *
```

**Exercice n°16**

En s'inspirant des codes précédents, réalisez les pyramides de nombres suivantes :

```
Donner le nombre de ligne: 5

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
Donner le nombre de ligne: 5

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

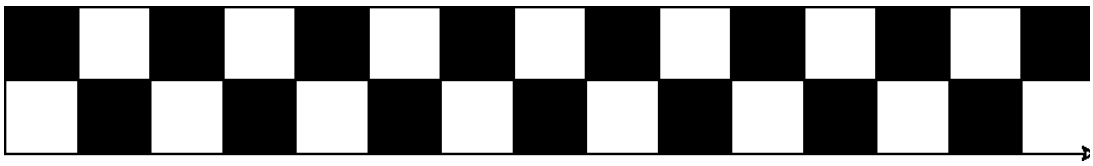
```
Donner le nombre de ligne: 6

1
2 3
3 4 5
4 5 6 7
5 6 7 8 9
6 7 8 9 10 11
```

# 6 Un dernier défi !

**Exercice n°17**
Sauras-tu écrire 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 !