Boucle for

Cours

Une boucle **for** en programmation est un mécanisme qui permet de répéter une série d'instructions un certain nombre de fois.

1. Boucle bornée simples

Répétition d'une instruction

Pour exécuter trois fois la même instruction, on avait jusqu'ici la possibilité de recopier cette instruction trois fois à la suite l'une de l'autre :

```
print("A")
print("A")
print("A")
```

Cette solution n'est pas raisonnable si le nombre de répétitions est important. De plus, elle n'est envisageable que si l'on connaît le nombre de répétitions lors de l'écriture du programme.

Python, comme de nombreux langages de programmation, propose une instruction appelée « boucle for » permettant de gérer cette répétition. En voici la forme la plus simple :

```
for _ in range(3) :
    print("A")
```

Le caractère souligné (_) fait partie de la construction.

On indique entre parenthèses après **range** un nombre de répétitions à effectuer, chaque répétition étant appelée un *tour*. L'instruction à répéter plusieurs fois est écrite après le symbole « : ». On l'appelle le *corps* de la boucle. Le nombre de Tours peut être donné par une expression arithmétique dont le résultat est calculé au préalable. En particulier, cette expression peut dépendre d'une variable.

Le programme suivant, récupère un nombre n saisi par l'utilisateur puis l'affiche un nombre de lignes égal à deux fois ce nombre n :

```
n = int(input())
for _ in range(2 * n):
    print("A")
```

Attention:

- Le nombre de tours de boucle doit être un entier
- Il est possible de prescrire un nombre de tours négatif. Dans ce cas il n'y aura aucun tour de boucle.
- La présence du symbole « : » entre la déclaration des répétitions et l'instruction est indispensable.

Répétition d'un bloc d'instruction

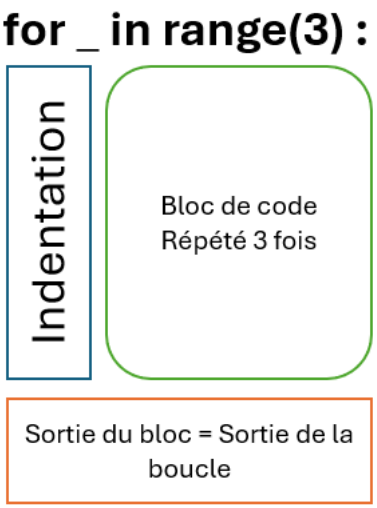
La répétition n'est pas limitée à une instruction, mais peut toucher une séquence.

Les instructions formant le corps de la boucle doivent être regroupées en un *bloc*, c'est à dire une suite de lignes en retrait du même nombre d'espaces. Ce nombre peut être arbitraire, mais l'usage est d'utiliser quatre espaces.

Exemple :

```
for _ in range(3) :
    print("Bonjour")
    print("_______")
print("Au revoir !")
```

En Python, l'indentation du code, c'est à dire les passages à la ligne et les retraits de chaque ligne, joue un rôle dans la signification du programme.



2. Utilisation de l'indice de boucle

A l'occasion d'une boucle bornée, il est possible d'introduire une nouvelle variable accessible à l'intérieur du corps de boucle et dont la valeur donne le numéro du tour en cours.

```
for i in range(10) :
    print(i)
```

On appelle cette variable spéciale l'*indice de boucle* ou le *compteur de boucle*.

- Le numéro du premier tour est 0, celui du deuxième est 1.
- Le numéro du dernier tour est donc un de moins que le nombre total de tours.

Chaque utilisation de **for** nécessite au moins une instruction à répéter.

3. Utilisation d'un accumulateur

Les blocs peuvent être composés avec toutes les instructions de Python, et notamment avec les instructions.

On peut ainsi à chaque tour de boucle récupérer de nouvelles entrées de l'utilisateur

Il est possible lors d'un tour de boucle de mettre à jour la valeur d'une variable, en repartant de la valeur qui avait été obtenu au tour précédent. Cette variable dont la valeur progresse à chaque tour de boucle est couramment appelée un *accumulateur*.

```
a = 1
for _ in range(4) :
    a = a + 2
print(a)
```

Le code précédent est équivalent au programme déplié suivant :

```
a = 1
a = a + 2
a = a + 2
a = a + 2
a = a + 2
print(a)
```

Tous deux affichent 9 après avoir successivement donné à la variable ***a*** les valeurs : 1, 3, 5, 7 et 9.

Une variable utilisée comme accumulateur dans une boucle **doit être initialisée avant la boucle**. Un code sans initialisation comme le suivant est erroné :

```
for _ in range(4) :
    a = a + 2
```

En effet, lorsque que commence le premier tour, la variable *a* n'a pas encore de valeur et la première exécution de l'instruction a = a +2 déclenchera l'erreur suivante :

```
Traceback (most recent call last):
  File "<string>", line 2, in <module>
NameError: name 'a' is not defined
```

Calcul de la moyenne

Les accumulateurs sont un bon outil dès lors qu'un programme effectue un grand calcul qui progresse un petit peu à chaque tour de boucle.

Exercice : Complétez le code Python ci-dessous, afin de calculer la moyenne des notes de *n* élèves.

```
n= int(input("Entrer le nombre d'élèves :"))
somme = ....
for _ in range(...) :
   k = int(input("Entrer une note : "))
   somme = .....
moyenne = .....
print("La moyenne est", moyenne)
```

Représentation de l'exécution

Voici par exemple un programme qui calcule la somme des carrés des *n* premiers entiers 0,1,2, … , n-1, le nombre *n* étant fourni par l'utilisateur.

Ligne	Code
1	s = 0
2	n = int(input("Combien d'entiers ? "))
3	for i in range(n):
4	z = i * i
5	s = s + z
6	print("Somme des carrés : ", s)

On montre ci-dessous un tableau représentant une exécution de ce programme dans laquelle l'utilisateur saisit pour *n* la valeur 4.

Exercice : Sans utiliser l'ordinateur, déduire à chaque étape, les valeurs des variables.

		s	n	i	z
Ligne 1		0			
Ligne 2		0	4		
Tour de boucle 1 (a)	3a				
	4a				
	5a				
Tour de boucle 2 (b)	3b				
	4b				
	5b				
Tour de boucle 3 (c)	3c				
	4c				
	5c				
Tour de boucle 4 (d)	3d				
	4d				
	5d				
Ligne 6					

Le compteur de boucle après la boucle

Une variable qui sert de compteur dans une boucle for a du sens pendant l'exécution de chaque tour de boucle. Cela dit, on peut remarquer que *i* existe encore après l'exécution de la boucle for et que cette variable contient la valeur correspondant au dernier tour de boucle. **On recommande de ne plus utiliser la valeur de *i* après la boucle.**

4. Jouer avec la numérotation

Jusqu'ici, on a utilisé l'opération range avec un unique paramètre décrivant le nombre de tours à effectuer.

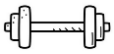
a) Il existe différentes variantes pour utiliser l'opération range

for i in range(**debut**, **fin**, **pas**) :

debut : La valeur que prendra le premier indice (indice de début)

fin : Les tours de boucle s'arrêtent juste avant que l'indice atteigne ou dépasse fin.

pas : Permet d'exprimer la valeur du saut d'indice entre chaque tour de boucle.

 **Activité :** Testez par vous-mêmes :

- **for i in range**(**0**, **10**, **1**) : print(i)
- **for i in range**(**-5**, **10**, **1**) : print(i)
- **for i in range**(**1**, **14**, **2**) : print(i)

Note :

Le cas **for i in range**(**n**) vu jusqu'ici est donc équivalent à **for i in range**(0, **n**, 1).

b) Il existe une dernière variante

for i in range(**debut**, **fin**) :

Dans cette variante, le pas sera toujours de 1.

5. Parcourir une chaîne de caractères

Voici un programme qui prend une chaîne de caractères et affiche chaque caractère un par un.

- Utilise une boucle for pour parcourir la chaîne de caractères donnée par l'utilisateur.
- Affiche chaque caractère sur une nouvelle ligne.

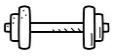
```
chaine = input("Entrez une chaîne de caractères : ")
for caractere in chaine:  # Parcours de chaque caractère de la chaîne
    print(caractere)  # Affiche chaque caractère sur une nouvelle ligne
```

Note :

La syntaxe est proche mais pas identique aux boucles for vues précédemment.

for caractere in chaine:

Ici caractere **n'est pas un indice** incrémenté à chaque tour de boucle, la variable caractere correspond à une **valeur**.

 **Activité :** Exercices 1,2, 3, 4 , 5 et 6

---

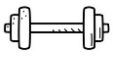
6. Les boucles imbriquées

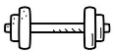
Le bloc d'instruction répété par une boucle for peut contenir n'importe quelle instruction Python légale, y compris ou autre boucle for :

```
for i in range(n) :
    print("Début de tour")
    for j in range(m):
        print(i, j)
    print("Fin de tour")
```

Exercice : Sans utiliser l'ordinateur, pour n = 4, et m = 3, déduire à chaque étape, les valeurs des variables.

Tour de boucle	Valeur(s) de i	Valeur(s) de j
Tour de boucle 0		
Tour de boucle 1		
Tour de boucle 2		
Tour de boucle 3		

 **Activité :** Exercices 7, 8 et 9.

 **Préparer une carte mentale afin de synthétiser la leçon.**