Cours n° 7 - Les Boucles itératives (instructions FOR et WHILE)

Boucle Pour : FOR

Lorsque l'on souhaite répéter un nombre **donné** de fois la même instruction ou le même bloc d'instructions, la commande **for … in range(1ère valeur, dernière valeur +1)** est la plus appropriée.
Admettons que l'on veuille afficher 5 fois le mot *bonjour*. Voici ce que l'on peut faire.

```
for i in range(5):        # pour i allant de 0 à 4
    print("bonjour")
```

→
```
bonjour
bonjour
bonjour
bonjour
bonjour
```

Par défaut : for i in range(5): est équivalent à : for i in range(0,5): et la boucle tournera 5 fois

→ i est un paramètre qui va prendre successivement les valeurs : $\{0;1;2;3;4\}$, soit $i \in 0;4$

→avec l'instruction for i in range(1,6): , la boucle tournera aussi 5 fois avec $i \in 1;5$

Subtilité : supposons que l'on veuille que variable i prenne successivement les valeurs 4,7,10,… jusqu'à 31 :

→on saisit : for i in range(4,32,3): le troisième paramètre définit le pas de variation

→par défaut, le pas vaut 1

→le pas doit être un nombre entier

Si on veut afficher les carrés des entiers de 1 à 7 :

```
for i in range(1, 8):          # pour i allant de 1 à 7
    print(i**2)
```

→
```
1
4
9
16
25
36
49
```

Puis terminons sur un exemple classique qui est le calcul de la somme des premiers entiers. Disons ici que l'on s'arrête à 30. Autrement dit, on veut calculer $1+2+3+...+30$ que l'on note aussi $\sum_{i=1}^{30} i$:

```
S = 0                          # on initialise la somme à 0
for i in range(1, 31):         # pour i allant de 1 à 30
    S = S + i                  # à chaque boucle FOR, on ajoute i à S
print("Somme =",S)
```

→ Somme = 465

Pour décrire le fonctionnement d'un programme, on utilise souvent un tableau décrivant l'évolution de chaque variable :

i	S
	0
1	1
2	3
3	6
…	…
30	465

Boucle Tant que : WHILE

Le principe de la boucle **while**, c'est d'exécuter un bloc d'instructions tant qu'une condition donnée est vraie.
On l'utilise lorsque l'on ne connait pas le nombre exact de boucles à réaliser.
Avec while, la variable n'augmente pas toute seule, il faut gérer sa progression après l'avoir initialisée.

Ex :
```
i = 1
while i <= 3:        # tant que i reste inférieur ou égal à 3, on reste dans la boucle
    print(i)
    i = i + 1
```
→
```
1
2
3
```

Ici tant que (while) la condition $(i \leq 3)$ est vraie, le bloc d'instructions (il y en a deux dans notre cas) est exécuté.
Regardons ce code pas à pas pour bien comprendre ce qu'il se passe.

```
1   i = 1
2   while i <= 3:
3       print(i)
4       i = i + 1
```

La deuxième instruction i = i + 1 est primordiale car elle assure que l'on va sortir de la boucle. En effet, ce qu'il faut éviter avec les boucles **tant que**, c'est de construire une **boucle sans fin**. C'est ce qui peut se produire si la condition du **while** n'est jamais vérifiée.
Pour savoir combien de fois la boucle **while** est exécutée, il peut être utile d'utiliser une variable qui servira de compteur, c'est-à-dire une variable (qu'il ne faut pas oublier d'initialiser) et incrémentée de 1 dans le bloc d'instructions.

L'ordre des écritures est primordial à l'intérieur de la boucle.
Comparons les écritures suivantes si l'on décide de calculer la somme des carrés des entiers successifs afin de trouver le plus petit entier pour lequel la somme des carrés dépasse 50 :

S = 0
i = 0
while S <= 50 :
  i += 1
  S += i**2
print(i,S)
 	
→ 5 55
	S = 0
	i = 1
	while S <= 50 :
	  S += i**2
	  i += 1
	print(i,S)
	
	→ 6 55

La bonne réponse est 5 car : $1^2+2^2+3^2+4^2+5^2=1+4+9+16+25=55$

→ seul le programme de gauche marche
→ sur celui de droite, la variable indique le rang suivant, il faudrait dans ce cas écrire :
print(i-1,S)

Boucle infinie : pour arrêter un programme qui tourne sans fin et consomme toute la mémoire :

Control K control KILL