Python 4 – **Boucles bornées en Python : for**

I – Boucle BORNEE

La boucle POUR/FOR est dite **bornée** car on peut connaître **à l'avance** le nombre de fois où elle va se dérouler.

II – Exactement pareil plusieurs fois

La syntaxe permettant de réaliser plusieurs fois la même chose est visible ligne 3 :

```
01 print("Avant boucle") ← pas tabulée
02
03 for _ in range(3): ← création de boucle
04     print("A")      ← tabulé
05     print("-- B")   ← tabulé
06
07 print("Après boucle") ← pas tabulé
```

Traduction de la ligne 3 : Fait 3 fois ces actions.

La **tabulation** (4 espaces) permet à l'interpréteur de savoir où se trouvent les instructions à réaliser en boucle.

On réalise la boucle autant de fois que la valeur précisée dans les parenthèses de la fonction range().

L'interpréteur Python exécute les lignes dans cet ordre :
L01
L03(1er tour)-L04-L05
L03(2e tour)-L04-L05
L03(2e tour)-L04-L05
L07

Il provoque cet affichage :

```
Avant boucle
A
-- B
A
-- B
A
-- B
Après boucle
```

III – Presque pareil plusieurs fois

On peut utiliser une **variable de boucle** dont la valeur varie à chaque tour de boucle pour réaliser presque la même chose. En notant **for v in range(4)**, la variable de boucle **v** va prendre des valeurs différentes à chaque tour de boucle : d'abord 0 puis 1 puis 2 puis 3.

ATTENTION : range(4) donc la dernière valeur est 3. On aura bien fait 4 tours : 0, 1, 2, 3.

```
01 print("Avant boucle")
02
03 for k in range(3):
04     print("A")
05     print(k)
06
07 print("Après boucle")
```

Traduction de la ligne 3 : Pour k variant de 0 à 3 progressivement.

L'interpréteur Python exécute les lignes dans cet ordre :
L01
L03(k=0)-L04-L05
L03(k=1)-L04-L05
L03(k=2)-L04-L05
L07

Il provoque cet affichage :

```
Avant boucle
A
0
A
1
A
2
Après boucle
```

IV – Problème typique : somme de 1 à n

La variable de boucle permet de faire la somme des entiers de 1 à 100 par exemple.

0 + 1 + 2 + 3 + 4 + 5 + 6 + … + 99 + 100.

```
01 s = 0                  # Somme s = 0 au début
02 for x in range(101):   # Pour x variant de 0 à 100
03     s = s + x          # Incrémente s de x
04 print(s)               # Affiche la somme s
```

```
L1
                s + x
L2(x=0)-L3): s = 0 + 0 donc 0
L2(x=1)-L3): s = 0 + 1 donc 1
L2(x=2)-L3): s = 1 + 2 donc 3
L2(x=3)-L3): s = 3 + 3 donc 6
L2(x=4)-L3): s = 6 + 4 donc 10
ect …

L4
```