I5 - Utiliser une boucle for

Variantes de syntaxes

Syntaxe numéro 1 : un seul nombre dans le range

```
for i in range(a):
    instructions
```

- ⋆ Nombre d'itérations : `a`
- ⋆ Intervalle de variation du compteur `i` : de `0` à `a-1`

Syntaxe numéro 2 : deux nombres dans le range

```
for i in range(a, b):
    instructions
```

- ⋆ Nombre d'itérations : `b-a`
- ⋆ Intervalle de variation du compteur `i` : de `a` à `b-1`

Dérouler une boucle

Pour comprendre ce que fait une boucle `for`, on doit « dérouler la boucle » c'est-à-dire étudier les opérations effectuées dans les premières étapes en calculant les différentes variables mises en jeu. Le principe est le suivant :

- ⋆ L'étape 0 est l'état des variables avant la boucle.
- ⋆ L'étape 1 est l'état des variables après la première exécution des instructions indentées.
- ⋆ L'étape 2 est l'état des variables après la deuxième exécution des instructions indentées.
- ⋆ Etc.

Par exemple pour la boucle suivante qui est souvent proposée lorsqu'on demande de calculer 2^{20} :

```
u = 2
for i in range(20):
    u = u * u
```

Étape	`i`	`u`
0		2
1	0	$2 \times 2 = 2^2 = 2^{2^1}$
2	1	$2^2 \times 2^2 = 2^4 = 2^{2^2}$
3	2	$2^4 \times 2^4 = 2^8 = 2^{2^3}$
$\vdots$	$\vdots$	$\vdots$
20	19	$2^{2^{20}}$

Il n'est pas nécessaire de remplir la dernière case : on constate assez vite que le nombre calculé ne sera pas celui qui était attendu. Pour obtenir le nombre calculé (celui dans la dernière case), il faut identifier ce qu'on appelle un invariant de boucle : il s'agit d'une propriété qui reste vraie à chaque étape. Dans cet exemple on a pour toute étape k, $u = 2^{2^k}$.

Un script correct qui permet de calculer 2^{20} est le suivant :

```
u = 1
for i in range(20):
    u = u * 2
print(u)
```

En effet, le déroulement de la boucle nous donne :

Étape	`i`	`u`
0		1
1	0	$1 \times 2 = 2^1$
2	1	$2 \times 2 = 2^2$
3	2	$2^2 \times 2 = 2^3$
$\vdots$	$\vdots$	$\vdots$
20	19	2^{20}

Calculer le n^{e} terme d'une suite récurrente

Suite récurrente simple

Soit $(u_n)_{n\geqslant 0}$ une suite définie par

$$u_0 = 1 \quad \text{et} \quad \forall n \in \mathbb{N}, \quad u_{n+1} = n - u_n$$

Pour construire la boucle permettant de calculer u_n, on peut commencer par construire le tableau du déroulement de la boucle que l'on attend :

Étape	`i`	`u`
0		$u_0 = 1$
1	0	$u_1 = 0 - u_0 = -1$
2	1	$u_2 = 1 - u_1 = 2$
3	2	$u_3 = 2 - u_2 = 0$
$\vdots$	$\vdots$	$\vdots$
$? = n$	$? = n-1$	$u_n = n - 1 - u_{n-1}$

Ceci permet d'identifier le nombre d'étape : ici ce sera n. Cela permet aussi de comprendre que la variable `i` va servir dans la formule d'actualisation de `u`. On a donc :

```
u = 1
for i in range(n):
    u = i - u            # Attention : u = n - u ne marche pas !
print(u)
```

Suite récurrente double

Soit $(v_n)_{n\geqslant 0}$ une suite définie par

$$v_0 = 1, v_1 = 2 \quad \text{et} \quad \forall n \in \mathbb{N}, \quad v_{n+2} = v_{n+1} + nv_n$$

Pour une telle suite, on aura besoin de travailler avec deux variables en même temps :

Étape	`i`	`a`	`b`
0		$v_0 = 1$	$v_1 = 2$
1	0	$v_1 = 2$	$v_2 = v_1 + 0 \times v_0 = 2$
2	1	$v_2 = 2$	$v_3 = v_2 + 1 \times v_1 = 4$
3	2	$v_3 = 4$	$v_4 = v_3 + 2 \times v_2 = 8$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$? = n-1$	$? = n-2$	v_{n-1}	$v_n = v_{n-1} + (n-2) \times v_{n-2}$
$? = n$	$? = n-1$	v_n	$v_{n+1} = v_n + (n-1) \times v_{n-1}$

Ici on a le choix :

- ⋆ si on affiche `a`, on doit faire `n` étapes :

```
a = 1
b = 2
for i in range(n):
    c = b + i * a
    a = b
    b = c
print(a)
```

- ⋆ si on affiche `b` (ou `c`), on doit faire `n-1` étapes (dans ce cas, il faut faire un cas particulier pour `n = 0` car `b` ne prend jamais la valeur de v_0) :

```
if n == 0:
    print(1)
else:
    a = 1
```

```
    b = 2
    for i in range(n-1):
        c = b + i * a
        a = b
        b = c
    print(b)                    # ou print(c)
```

Calculer une somme et un produit

Calculer une somme

Pour calculer la somme $\sum_{k=a}^{b} u_k = u_a + u_{a+1} + \cdots + u_b$:

- ⋆ on initialise `S` à `0` (**on initialise toujours une somme à** `0`)
- ⋆ on fait une boucle `for k in range(a, b+1)` afin que `k` varie de `a` à `b+1-1 = b`

Par exemple, pour calculer la somme $\sum_{k=10}^{100} \frac{k+1}{k^2}$:

```
S = 0
for k in range(10, 101):
    S = S + (k+1) / (k**2)  # ou bien S += (k+1) / (k**2)
print(S)
```

Calculer un produit

Pour calculer le produit $\prod_{k=a}^{b} u_k = u_a \times u_{a+1} \times \cdots \times u_b$:

- ⋆ on initialise `P` à `1` (**on initialise toujours un produit à** `1`)
- ⋆ on fait une boucle `for k in range(a, b+1)` afin que `k` varie de `a` à `b+1-1 = b`

Par exemple, pour calculer le produit $\prod_{k=10}^{100} \frac{k+1}{k^2}$:

```
P = 1
for k in range(10, 101):
    P = P * (k+1) / (k**2)  # ou bien P *= (k+1) / (k**2)
print(P)
```