Boucles bornées

8.1

Précisions sur la boucle for

8.1.1 Même action voir Intro 1.4.9

Avec `for _ in range(10)`, on peut réaliser 10 fois exactement la même action.

8.1.2 Presque même action voir Intro 1.4.10

Avec `for k in range(10)`, on peut réaliser 10 fois presque la même action en utilisant la valeur successive de la variable de boucle k : 0 puis 1 puis 2... jusqu'à 9.

8.1.3 Utilisation de range()

Visualiser range()

L'évaluation de `range(5)` n'est un tableau mais on peut représenter la réponse sous forme d'un tableau à l'aide de la fonction native `list()`.

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

Un seul argument

On précise alors la **borne finale exclue**.

Lorsqu'on utilise range(10), l'interpréteur Python comprendra que :

— on veut commencer à 0 ;
— la valeur "presque" finale (c'est à dire exclue) est 10 ;
— on incrémentera la variable de boucle de +1 à chaque tour de boucle.

```
>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> for k in range(10):
    print(k)
```

0	4	8
1	5	9
2	6	
3	7	

Deux arguments

On précise alors la **valeur initiale** puis la **borne finale exclue**.

Lorsqu'on utilise range(3, 10), l'interpréteur Python comprendra que :

— on veut commencer à 3 ;
— la valeur "presque" finale (c'est à dire exclue) est 10 ;
— on incrémentera la variable de boucle de +1 à chaque tour de boucle.

```
>> list(range(3, 10))
[3, 4, 5, 6, 7, 8, 9]
```

```
>>> for k in range(3, 10):
    print(k)
```

3	6	9
4	7	
5	8	

Trois arguments

On précise alors la **valeur initiale**, la **borne finale exclue** et la valeur du pas : combien gagne-t-on ou perd-t-on à chaque tour de boucle.

Lorsqu'on utilise range(3, 10, 2), l'interpréteur Python comprendra que :

— on veut commencer à 3 ;
— la valeur "presque" finale (c'est à dire exclue) est 10 ;
— on incrémentera la variable de boucle de +2 à chaque tour de boucle.

```
>> list(range(3, 10, 2))
[3, 5, 7, 9]
```

```
>>> for k in range(3, 10, 2):
    print(k)
```

3	7	
5	9	

On peut fournir un pas négatif. Ca complique la compréhension de la vraie valeur finale : si la valeur finale est 1 en décroissant, la dernière valeur disponible est donc... 2.

small

```
>>> list(range(10, 1, -3))
[10, 7, 4]
```

```
>>> for k in range(10, 1, -3):
    print(k)
```

10	7	4

8.1.4 Boucle POUR : boucle BORNEE

Les boucle POUR sont des boucles bornées : on ne peut pas boucler à l'infini puisqu'on pourrait prévoir à l'avance combien de fois le bouclage sera réalisé.

8.2

Sommes et concaténations successives

8.2.1 somme avec additions successives

Nous voudrions faire la somme de plusieurs nombres 0 + 1 + 2 + 3 + 4 + 5 + 6 ... + 1000 jusqu'à un entier final au choix (1001 sur l'exemple).

Nous allons calculer la somme par additions successives, en écrasant la version précédente par la nouvelle version.

La signature de l'addition d'entiers est

`int + int -> int`

Cela reviendrait à faire ceci à la main (et jusqu'à 1000, ça risque d'être long à taper...) :

```
somme = 0
somme = somme + 1  # donc 1
somme = somme + 2  # donc 3
somme = somme + 3  # donc 6
somme = somme + 4  # donc 10
...
somme = somme + 1000
```

Plutôt que de tout faire à la main, utilisons une boucle dont la variable de boucle se nommerait **`nombre`**.

```
# Initialisation du compteur
somme = 0

# Pour chaque nombre entier de 1 à 1000
for nombre in range(1, 1001):
    # Incrémente somme avec ce nombre
    somme = somme + nombre
```

8.2.2 concaténations successives

Nous voudrions créer une chaîne de caractères contentant "0 1 2 3 4 5 6 ... 1000" jusqu'à un entier final au choix (1000 sur l'exemple).

La signature de la concaténation de strings est

`str + str -> str`

Cela reviendrait à faire ceci à la main (et jusqu'à 1000, ça risque d'être long à taper...) :

```
chaine = ""
chaine = chaine + str(0) + " "  # "0 "
chaine = chaine + str(1) + " "  # "0 1 "
chaine = chaine + str(2) + " "  # "0 1 2"
chaine = chaine + str(3) + " "  # "0 1 2 3 "
...
chaine = chaine + str(1000) + " "
```

Plutôt que de tout faire à la main, utilisons une boucle dont la variable de boucle se nommerait **`nombre`**.

```
# Initialisation du string
chaine = ""

# Pour chaque nombre entier de 0 à 1000
for nombre in range(1001):
    # Concatène ce nombre et un espace à chaine
    chaine = chaine + str(nombre) + " "
```

Attention à la phase d'initialisation de la ligne 1. Elle est très importante et nous allons la retrouver très souvent cette année.

8.3

Mode DEBUG de Thonny

Vous devez être capable de faire fonctionner le mode debug de Thonny, notamment de savoir activer l'instruction suivante et le fait de rentrer dans les boucles.

8.4

DEBUG avec Pythontutor

Vous devez être capable de faire fonctionner le mode debug d'un programme que vous copiez sur le site **`https://pythontutor.com/`**