On découvre

- comment répéter l'exécution de certaines parties d'un programme tant qu'une condition est respectée
- ce qu'est une itération conditionnelle

Les boucles `for` permettrent de répéter une séquence d'instructions sur un itérable, donc le nombre de répétitions est connu à l'avance.

Tandis que les boucles `while` permettent de répéter une séquence d'instructions tant qu'une certaine condition est – ou n'est pas – vérifiée. Le nombre d'itérations n'est alors pas connu à l'avance.

Une telle structure est une boucle appelée *itération conditionnelle*, encore appelée *boucle tant que*.

1 Syntaxe et flot d'exécution

La syntaxe de la boucle tant que utilise le mot-clé `while` :

```
...
while <condition> :
    <instructions>
...
```

Le flot d'exécution correspondant peut être illustré par le graphe :

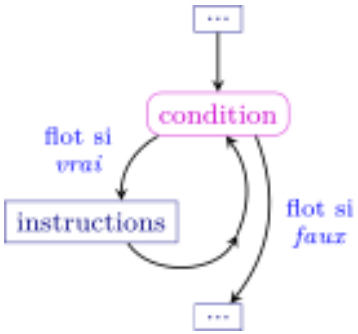

Figure 1: flot while

- la condition est évaluée. On obtient une valeur booléenne.
- si cette valeur est `True`
  - les instructions associées au `while` sont exécutées
  - on retourne au point précédent pour évaluer à nouveau la condition
- si cette valeur est `False`
  - on passe à l'exécution des instructions suivantes (`...`).

2 Un exemple

On souhaite déterminer à partir de quelle valeur de n la somme des premiers entiers $1 + 2 + 3 + \cdots + n$ dépasse un seuil donné, par ex 6.

On va effectuer le calcul de la somme tant que celle-ci ne dépasse pas 6. Quand la somme dépasse 6, on a trouvé la valeur de n qu'on souhaitait calculer. On ne sait pas a priori combien d'additions on devra effectuer.

Il nous faut pouvoir identifier :

- la condition pour continuer la boucle : la somme est inférieure ou égale à 6 (ou, de manière duale, la condition pour arrêter la boucle : la somme dépasse 6)
- la ou les instructions à répéter : construction des premiers entiers et calcul de la somme de ces entiers

On écrit donc la fonction suivante :

```
def entier_tq_somme_depasse_seuil(seuil:int) -> int:
    """ Renvoie l'entier n à partir duquel la somme des entiers
    de 1 à n dépasse (strictement) `seuil`.

    Précondition : /
    Exemple(s) :
    $$$ entier_tq_somme_depasse_seuil(6)
    4
    $$$ entier_tq_somme_depasse_seuil(-2)
    1
    """
    n = 1                                   #1
    somme = 1                               #2
    while somme <= seuil:                   #3
        n = n + 1                           #4
        somme = somme + n                   #5
    return n
```

2.1 Exemple d'exécution avec 3 itérations

Voici comment se déroule l'appel `entier_tq_somme_depasse_seuil(6)`.

Après exécution des lignes 1 et 2 (lignes d'initialisation), la mémoire contient :

`seuil`	6
`n`	1
`somme`	1

Ensuite (ligne 3) Python évalue la condition `somme <= seuil` qui vaut `True`. On effectue donc une première itération, en exécutant le corps de la boucle (lignes 4 et 5).

À la fin de cette première itération, la mémoire contient donc :

`seuil`	6
`n`	2
`somme`	3

Python évalue à nouveau la condition `somme <= seuil` qui vaut `True` (ligne 3). On effectue donc une deuxième itération, en exécutant le corps de la boucle (lignes 4 et 5).

À la fin de cette deuxième itération, la mémoire contient donc :

`seuil`	6
`n`	3
`somme`	6

Python évalue à nouveau la condition `somme <= seuil` qui vaut `True` (ligne 3). On effectue donc une troisième itération, à la fin de laquelle la mémoire contient :

seuil	6
n	4
somme	10

Python évalue à nouveau la condition `somme <= seuil` qui vaut `False` (ligne 3).

Il n'y a donc pas d'autre itération, la boucle `while` termine et la fonction renvoie la valeur de `n`, c'est-a-dire `4`.

2.2 Exécution sans itération

Voici comment se déroule l'appel `entier_tq_somme_depasse_seuil(-2)`.

Après exécution des lignes 1 et 2 (lignes d'initialisation), la mémoire contient :

seuil	-2
n	1
somme	1

Ensuite (ligne 3) Python évalue la condition `somme <= seuil` qui vaut `False`.

Il n'y a donc pas d'itération, la boucle `while` termine et la fonction renvoie la valeur de `n`, c'est-à-dire `1`.

3 À propos des boucles `while`

3.1 Évolution de la valeur de la condition

D'une manière générale un code utilisant une boucle `while` a la forme suivante :

```
# (1) initialisations
while <condition> :
    # (2)
    <instructions>
    # (3)
# (4)
```

dans lequel les commentaires `(1)`, `(2)`, `(3)` et `(4)` servent à indiquer un emplacement dans le flot d'exécution.

1. Juste avant la boucle – au point `(1)` –, l'expression booléenne `condition` peut prendre les valeurs
   - `True`, auquel cas on est sûr qu'il y aura au moins une itération, ou
   - `False`, auquel cas on est sûr qu'il n'y aura aucune itération.
2. Au début d'une itération – au point `(2)` –, l'expression booléenne `condition` est nécessairement vraie.
3. À la fin d'une itération – au point `(3)`–, l'expression booléenne `condition` peut valoir
   - `True`, auquel cas la boucle continuera pour une autre itération au moins, ou
   - `False`, auquel cas, la boucle s'arrêtera.
4. Juste après la boucle – au point `(4)` –, l'expression booléenne `condition` est nécessairement fausse. Sinon la boule n'aurait pas terminé.

3.2 Boucle infinie

S la condition de la boucle `while` est toujours vraie, la boucle ne termine pas[1]. On parle de *boucle infinie*.

Les boucles infinies proviennent souvent d'une erreur dans le code.

[1]Symptôme dans Thonny : l'interpréteur ne renvoie pas la main (dans le cas où on a exécuté un programme ou entré une commande dans l'interpréteur), ou la fenêtre de L1Test reste blanche (dans le cas où on a exécuté un test qui déclenche une boucle infinie). Il faut alors cliquer sur le bouton `Stop` pour interrompre l'exécution.

Par exemple, supposons que dans la fonction précédente on a par erreur écrit `n = n - 1` au lieu de `n = n + 1` (ligne 6).

Dans ce cas l'appel `entier_tq_somme_depasse_seuil(6)` déclenche une boucle infinie. En effet `n` décroit et n'atteindra jamais `6`.

Une erreur classique est de parcourir les indices `i` d'une séquence en initialisant `i` à 0 et en oubliant d'incrémenter `i` dans la boucle, habitués que nous sommes à écrire une boucle `for` qui gère l'avancée de la variable d'itération pour nous.

4 Memento

- l'instruction d'itération conditionnelle `while` permet d'exécuter un bloc d'instructions tant qu'une condition est vérifiée