La boucle `while`

Complément - niveau basique

Boucles sans fin - `break`

Utiliser `while` plutôt que `for` est une affaire de style et d'habitude. Cela dit avec l'apparition des itérateurs, l'usage du `for` est en général privilégié pour les boucles finies et déterministes.

Le `while` reste malgré tout d'un usage courant avec une condition `True`.

Par exemple le code de l'interpreteur interactif de python pourrait ressembler, vu de très loin, à quelque chose comme ceci

```
while True:
    print(eval(read()))
```

Notez bien par ailleurs que les instructions `break` et `continue` fonctionnent, à l'intérieur d'une boucle `while`, exactement comme dans un `for`.

Complément - niveau intermédiaire

Rappel sur les conditions

On peut utiliser dans une boucle `while` toutes les formes de conditions que l'on avait vues à l'occasion de l'instruction `if`.

Dans le contexte de la boucle `while` on comprend mieux, toutefois, pourquoi le langage autorise d'écrire des conditions dont le résultat n'est **pas nécessairement un booléen**. Voyons cela sur un exemple simple:

```
# une autre façon de parcourir une liste
liste = ['a', 'b', 'c']
while liste:
    element = liste.pop()
    print element
```

Il peut être intéressant de comparer ce code avec ce qu'on obtiendrait avec une simple boucle `for`.

- Une première différence est qu'avec `while` on pourrait facilement traiter les éléments de la liste deux par deux si cela faisait du sens dans le contexte.

- On voit aussi tout de suite que les éléments sont traités en sens inverse; il faut savoir qu'avec le type `list` l'opération `pop()` est très efficace, car elle fonctionne en temps constant – ce qui d'ailleurs n'est pas le cas de `pop(0)`. Aussi la boucle `while` peut être plus efficace, pour traiter une liste en sens inverse.

- Mais contrairement à une boucle `for` cette forme est intrusive, en ce sens que la liste est modifiée; alors que, on le rappelle, à l'intérieur d'une boucle `for` on ne **doit pas** modifier l'objet de la boucle.

Une curiosité : la clause `else`

Signalons enfin que la boucle `while` – au même titre d'ailleurs que la boucle `for`, peut être assortie d'une clause `else` (https://docs.python.org/2/reference/compound_stmts.html#the-while- statement) , qui est exécutée à la fin de la boucle, sauf dans le cas d'une sortie avec `break`.

```
# Un exemple de `else` dans un while

def scan(liste, break_mode):
    # si break_mode est vrai on va faire un break après le premier élément de la l
iste
    message = "avec" if break_mode else "sans"
    print 20*'-', "\nScan {} break".format(message)
    while liste:
        print liste.pop()
        if break_mode:
            break
    else:
        print 'else...'


scan(['a'], False)

scan(['a'], True)
```

Ce trait est toutefois **très rarement** utilisé.