Leçon d'informatique : Boucle while

S. Benlhajlahsen - PCSI$_1$

Sommaire

Idée Si on peut itérer un processus jusqu'à satisfaire une condition (par exemple une précision numérique souhaitée), on ne sait pas forcément combien de tours de boucles on va devoir faire. Dans ces conditions, la boucle `for` n'est pas adaptée, on lui préfèrera une structure plus générale : la boucle `while`.

I Présentation

I.A Synthaxe

Synthaxe La syntaxe fait intervenir le mot clef `while` et une clause à valeur booléenne (voir figure 2) que l'on peut résumer par : **tant que** la clause vaut `True` ; la boucle continue ses itérations.

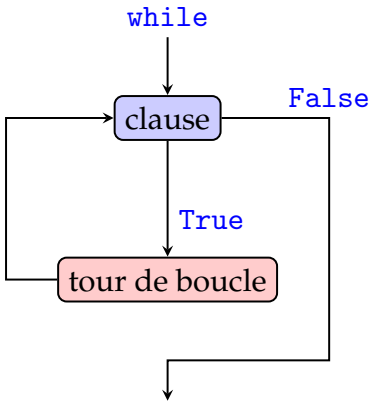

FIGURE 1 – Shéma de principe de la boucle `while`.

```
while clause :
    instructions
```

FIGURE 2 – Synthaxe de la boucle `while`.

Remarque Il n'y a pas de variable de boucle cette fois, mais on peut en créer une manuellement si besoin, par exemple en l'initialisant à zéro avant la boucle et en lui ajoutant 1 à chaque tour de boucle. Voir exemples plus bas.

I.B Dangers de la boucle infinie

Boucle infinie Une cause d'erreur très classique est que la clause ne puisse jamais devenir égale à False, de sorte que la boucle ne s'arrêtera jamais ("boucle infinie"). Il faudra alors utiliser votre IDE pour interrompre en force l'exécution du programme.

À retenir : Quand vous utilisez une boucle `while`, assurez-vous que la clause peut réellement devenir égale à `False`.

II Exemples

II.A Exemple trivial

```
1 i = 0                          # initialisation de la variable
2 while i<10:
3     print(i)
4     i+=1                       # actualisation de la variable
```

FIGURE 3 – Exemple pour la boucle `while`.

Exemple de la figure 3 Si on exécute cet exemple, la console affiche successivement les entiers 0 à 9. Dans cet exemple, `i` joue le rôle de **compteur de boucle**. On constate alors que :

- en ligne 1, on a créé ce compteur et on lui a donné une valeur initiale ;
- en ligne 4, on actualise le compteur à chaque tour de boucle.

C'est cette actualisation[a] qui va permettre l'arrêt de la boucle.

a. souvent oubliée!!!

Exemple de la figure 4 Le compteur de boucle peut être implicite ou explicite. Dans l'exemple de la figure 4, le compteur de la boucle est implicitement la taille de la liste `L`. L'affichage de la ligne 4 donne alors `[0,0,0,0]`.

```
1 L = []
2 while len(L)<= 3:
3     L.append(0)
4 print(L)
```

FIGURE 4 – Exemple pour la boucle `while`.

II.B Test de primalité

Remarque On a déjà écrit un test de primalité. On ne sait pas réellement combien de tours de boucle vont être nécessaires. On connaît seulement le nombre maximal de tours, et on interrompt la boucle avec un `return` si nécessaire. Réécrivons cet algorithme avec une boucle `while` (voir figure 5). La logique de base n'a pas vraiment changé, mais on note la nécessité de créer soi-même la variable de boucle `k` et l'intervention d'un booléen `res` :

- On parcourt tous les entiers de 2 à $\sqrt{n}$ pour trouver un diviseur (k est initialisée à 2)
- `res` sert de "variable marqueur" : on suppose que `n` est premier (variable égale à `True`) et on change d'avis (passage à `False`) si on trouve un diviseur.

La boucle continue **tant que** deux conditions sont simultanément (`and`) satisfaites :

- `k` n'a pas atteint sa valeur maximale $\sqrt{n}$.
- `res` vaut encore `True` (on n'a pas encore trouvé de diviseur).

Il est logiquement équivalent de dire que la boucle s'arrête si l'une des deux conditions n'est plus satisfaite[a].

a. Voir cours de mathématiques : la négation de "A et B" est "(non A) ou (non B)").

```python
import math as m
def test_primalite(n) :
    if n==2 or n==3:
        res = True
    racine = round(m.sqrt(n))
    k = 2
    res = True
    while k <= racine and res :
        if n%k == 0 :
            res = False
        k += 1
    return res
```

FIGURE 5 – Test de primalité.

II.C Écriture binaire d'un entier

Idée : Soit n un entier. L'écriture binaire de cet entier est de la forme :

$$n = a_{p-1}2^{p-1} + a_{p-2}2^{p-2} + \cdots + a_1 2^1 + a_0 2^0$$

où p est l'entier tel que $2^{p-1} \leqslant n < 2^p$. C'est aussi le nombre de chiffre dans l'écriture en binaire. Par exemple, 9 s'écrit 1001_2 en écriture binaire et constate bien que $2^3 \leqslant 9 < 2^4$. Etant donné un entier n, la recherche de p peut se faire à l'aide d'une boucle `while` (voir figure 6).
On peut ensuite renvoyer ensuite l'écriture binaire sous forme de liste ou de chaîne de caractères. La méthode consiste à diviser n par 2^{p-1}, puis de diviser le reste par 2^{p-2} jusqu'à 2^0 (voir figure 7).

```python
def puissance(n):
    p = 0
    while n>=2**p:
        p+=1
    return p
```

FIGURE 6 – Détermination du plus petit entier tel que $2^{p-1} \leqslant n < 2^p$.

```python
def binaire1(n):
    p = puissance(n)
    L = []
    for i in range(p-1,-1,-1):
        q = n//(2**i)
        n = n%(2**i)
        L.append(q)
    return L
print(binaire1(9)) # renvoie [1,0,0,1]
```

FIGURE 7 – Écriture binaire sous forme de liste.

```python
def binaire2(n):
    p = puissance(n)
    res = ""
    for i in range(p-1,-1,-1):
        q = n//(2**i)
        n = n%(2**i)
        res += str(q)
    return res
print(binaire2(9)) # renvoie "1001"
```

FIGURE 8 – Écriture binaire sous forme de chaîne de caractères.

III Exercices

Exercice 1 : Écrivez une fonction `liste_chiffres(nombre)` recevant en entrée un nombre entier positif et renvoyant en sortie la liste de ses chiffres. De nombreuses méthodes sont possibles, utilisez la plus classique (boucle while avec des divisions par 10 successives).

```python
# méthode boucle while
def puissance10(nombre):
    # renvoie le plus petit entier k tel que nombre < 10**k
    k = 0
    while nombre > 10**k:
        k+=1
    return k
print(puissance10(1234))

def liste_chiffres(nombre):
    res = []
    k = puissance10(nombre)
    while k>0:
        res.append(nombre//10**(k-1))
        nombre = nombre%10**(k-1)
        k -=1
    return res
print(liste_chiffres(1457))

# autre méthode : utilisant les chaînes
def liste_chiffres(nombre):
    x =  str(nombre)
    res = []
    for i in x:
        res.append(int(i))
    return res
print(liste_chiffres(1457))
```

FIGURE 9 – Réponse à l'exercice 1.