

Boucle while

Informatique pour tous

nb zero n !

Limitation de la boucle for

Imaginons que l'on souhaite faire un programme qui demande l'âge de l'utilisateur.

On peut le faire en écrivant :

```
age = int(input("Quel est votre age?"))
```

Limitation de la boucle for

Cependant, pour éviter les fautes de frappes, on voudrait éviter que l'utilisateur rentre un nombre négatif.

On veut donc redemander l'âge **tant que** le nombre rentré est négatif.

Limitation de la boucle for

On ne peut pas utiliser de boucle for, car on ne sait pas à l'avance combien de fois on va devoir demander l'âge.

Limitation de la boucle for

On ne peut pas utiliser de boucle for, car on ne sait pas à l'avance combien de fois on va devoir demander l'âge.

On utilise alors une boucle **while** :

```
age = int(input("Quel est votre age?"))  
while age < 0:  
    print("Veuillez entrer un nombre positif...")  
    age = int(input("Quel est votre age?"))
```

Boucle while

Syntaxe générale de la boucle `while` s'exécutant **tant que** condition est évaluée à `True` :

```
# code avant le while
while condition:
    # instructions
    # autres instructions
# code après le while
```

Boucle while

```
# code avant le while
while condition:
    # instructions
    # autres instructions
# code après le while
```

condition peut être, par exemple :

- Une variable booléenne.
- Une comparaison : `n < 10`, `chaine == "blabla"`.
- Construit avec `and`, `or`, `not` : `x >= 0 and x <= 10`.

Boucle while

Toute boucle for peut se transformer en boucle while :

```
for i in range(n):  
    ...
```

```
i = 0  
while i < n:  
    i = i + 1  
    ...
```


Boucle while

Toute boucle for peut se transformer en boucle while :

```
for i in range(n):  
    ...
```

```
i = 0  
while i < n:  
    i = i + 1  
    ...
```

On utilisera une boucle for quand c'est possible, pour sa plus grande simplicité.

Question

Comment transformer la boucle for suivante en boucle while équivalente ?

```
for i in range(m, n, p):  
    ...
```

Question

Comment transformer la boucle for suivante en boucle while équivalente ?

```
for i in range(m, n, p):  
    ...
```

```
i = m  
while i < n:  
    i = i + p  
    ...
```

⚠ Contrairement aux boucles `for`, il est facile d'écrire une boucle `while` qui ne termine pas (on parle aussi de boucle infinie) !

⚠ Contrairement aux boucles `for`, il est facile d'écrire une boucle `while` qui ne termine pas (on parle aussi de boucle infinie) !

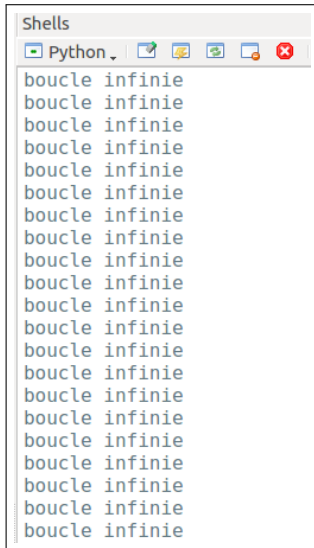
```
while True:  
    print("boucle infinie")
```

⚠ Contrairement aux boucles `for`, il est facile d'écrire une boucle `while` qui ne termine pas (on parle aussi de boucle infinie) !

```
while True:  
    print("boucle infinie")
```

Il faudra donc faire attention à ce que notre boucle termine, quand on utilise `while`.

Boucle while



```
Shells
Python
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
boucle infinie
```

Question

Écrire une boucle `while` permettant de trouver la plus petite puissance de 2 dépassant 1 million.

Question

Écrire une boucle `while` permettant de trouver la plus petite puissance de 2 dépassant 1 million.

```
n = 1
while n < 10**6:
    n = 2 * n
print("La lère puissance de 2 dépassant 1 million est")
print(n)
```

Recherche dichotomique

Soit L une liste de nombres. On a vu une fonction `contient` permettant de savoir si un élément e appartient à L en au plus $\text{len}(L)$ itérations.

Recherche dichotomique

Soit L une liste de nombres. On a vu une fonction `contient` permettant de savoir si un élément e appartient à L en au plus $\text{len}(L)$ itérations.

```
def contient(L, e):  
    for i in range(len(L)):  
        if L[i] == e:  
            return True  
    return False
```

Si L est **triée**, on va voir qu'on peut savoir si e est dans L plus rapidement, en comparant e avec le **milieu** m de L :

- Si $e == m$, on a trouvé notre élément.
- Si $e > m$, il faut chercher e dans la partie droite de L
- Si $e < m$, il faut chercher e dans la partie gauche de L

Recherche dichotomique

Exemple : on veut savoir si 14 appartient à la liste :

$$L = [-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]$$

Le nombre d'itérations de la fonction `contient(L, 14)` est :

Exemple : on veut savoir si 14 appartient à la liste :

$$L = [-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]$$

Le nombre d'itérations de la fonction `contient(L, 14)` est : 11.

Avec la recherche dichotomique :

`[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]`

`9 < 14`

Recherche dichotomique

Avec la recherche dichotomique :

`[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]`

$9 < 14$

Recherche dichotomique

Avec la recherche dichotomique :

`[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]`

$15 > 14$

Avec la recherche dichotomique :

$[-2, 1, 2, 4, 6, 7, 8, 9, \boxed{11, \underline{12}, 14}, 15, 18, 22, 54]$

$12 < 14$

Avec la recherche dichotomique :

[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, **14**, 15, 18, 22, 54]

14 trouvé !

Recherche dichotomique

Avec la recherche dichotomique :

`[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]`

14 trouvé !

On a fait seulement 4 itérations.

Recherche dichotomique en Python

```
def dichotomie(L, e):  
    debut = 0 # indice de début  
    fin = len(L) # indice de fin exclu  
    while debut < fin:  
        milieu = (debut + fin) // 2  
        if L[milieu] == e:  
            return True  
        elif L[milieu] < e: # il faut chercher à droite  
            debut = milieu + 1  
        else: # il faut chercher à gauche  
            fin = milieu  
    return False
```

Question

Combien de fois la boucle `while` s'exécute pour une liste de taille $n = \text{len}(L)$?

Question

Combien de fois la boucle `while` s'exécute pour une liste de taille $n = \text{len}(L)$?

A chaque exécution, on divise (environ) par deux la zone de recherche. Au bout de deux exécutions, elle sera divisée par 4, puis 8, ... et 2^p au bout de p exécutions.

Question

Combien de fois la boucle `while` s'exécute pour une liste de taille $n = \text{len}(L)$?

A chaque exécution, on divise (environ) par deux la zone de recherche. Au bout de deux exécutions, elle sera divisée par 4, puis 8, ... et 2^p au bout de p exécutions.

Donc, au bout de p exécutions, le nombre d'éléments de la zone de recherche est environ :

$$\frac{n}{2^p}$$

Question

Combien de fois la boucle `while` pour une liste de taille $n = \text{len}(L)$?

Au bout de p exécutions, le nombre d'éléments de la zone de recherche est environ :

$$\frac{n}{2^p}$$

Quand $\frac{n}{2^p} = 1$, c'est à dire $p = \log_2(n)$, la zone de recherche n'a plus que 1 élément et la fonction s'arrête.

Question

Combien de fois la boucle `while` pour une liste de taille $n = \text{len}(L)$?

Au bout de p exécutions, le nombre d'éléments de la zone de recherche est environ :

$$\frac{n}{2^p}$$

Quand $\frac{n}{2^p} = 1$, c'est à dire $p = \log_2(n)$, la zone de recherche n'a plus que 1 élément et la fonction s'arrête.

Donc la boucle `while` s'exécute environ $\log_2(n)$ fois, ce qui est beaucoup plus rapide que les n exécutions de la fonction `contient` !