

La boucle non bornée

Définition

Une boucle non bornée consiste en une série d'instructions qui sont exécutées en boucle . Cette , dite de , est exprimée par un **booléen** : tant que ce booléen a pour valeur `True`, les instructions dans la boucle sont exécutées. Quand le booléen prend la valeur `False`, le programme exécute les instructions restantes dans l'itération de boucle courante, puis continue son exécution **après** la boucle.

Une boucle non bornée s'écrit en Python à l'aide du mot-clef `while`.

La syntaxe Python est similaire à celle utilisée pour les instructions conditionnelles :

- la condition après le `while` est suivie de deux points `:"` ;
- les instructions sont placées à une indentation sur la droite relativement au `while`.

Exemple :

Prenons l'exemple de la conversion d'un nombre `n` de la base 10 à la base 2. On fait des divisions euclidiennes successives par 2 pour obtenir `n` écrit en binaire.

```
1 n = 4
2 r = n % 2    # on calcule le reste de la division euclidienne de n par 2
3 n = n // 2   # on remplace n par son quotient dans la division par 2
4 r = n % 2    # on calcule le nouveau reste
5 n = n // 2   # on remplace n à nouveau
6 r = n % 2
7 n = n // 2
```

Compléter la table de trace correspondant à l'exécution de ce code :

N° de ligne	n	r
1		
2		
3		
4		
5		
6		
7		

On répète les mêmes opérations, **jusqu'à ce que** `n` vale 0. Au lieu de réécrire les mêmes instructions, on peut utiliser une boucle non bornée comme suit.

```
1 n = 4
2 while n!=0:
3     r = n % 2
4     n = n // 2
```

→ On peut ajouter des appels à la fonction `print` dans ces scripts afin de vérifier les résultats.

C'est un défaut de conception qui doit être absolument évité : pour cela, on doit s'assurer de modifier dans la boucle les variables qui interviennent dans la condition de continuation.

Exercice 1 – (Sur papier)

```
1  compteur = 1
2  while compteur < 10:
3      compteur = compteur * 2
4  print(compteur)
```

1. Remplir la table de trace suivante, avec le nombre d'itération de boucle adéquat.

N° de ligne	condition	compteur
1		
2		
3		
2		
3		

2. En déduire la valeur affichée à la fin de l'exécution du programme.

Exercice 2 – (Sur papier puis à vérifier sur ordinateur)

Écrire un programme remplissant les conditions suivantes :

- initialiser une variable `n` à la valeur 0 ;
- augmenter (**incrémenter**) la valeur de `n` de 1 à chaque passage dans une boucle `while` ;
- la valeur de `n` à la fin de l'exécution de la boucle doit être de 5.

Exercice 3 – (Sur papier puis à vérifier sur ordinateur)

L'accès à une page Web reste verrouillé tant que l'on n'entre pas le bon mot de passe. On donne le programme suivant (incomplet), qui gère cet accès.

```
1  mdp = "hj51dpM@"
2  rep = ""
3
4  while .....:
5      rep = input("Entrer le mot de passe pour accéder à la page.")
6
7  print("Accès autorisé")
```

La fonction `input` affiche un message et demande à l'utilisateur d'entrer une valeur qui est récupérée dans une variable, ici `rep`.

1. Identifier les différentes variables du programme et indiquer leur utilité.
2. Compléter le programme avec la condition qui convient.

Exercice 4 – (Sur papier puis à vérifier sur ordinateur)

On considère les scripts ci-dessous.

1 <code># script 1</code>	8 <code>n = 0</code>	14 <code>n = 0</code>
2 <code>n = 5</code>	9 <code>a = 1</code>	15 <code>a = 1</code>
3 <code>s = 0</code>	10 <code>while n < 8:</code>	16 <code>while a < 1000:</code>
4 <code>while n > 0:</code>	11 <code> n = n + 1</code>	17 <code> a = a * 2</code>
5 <code> n = n - 1</code>	12 <code> a = a * 2</code>	18 <code> n = n + 1</code>
6 <code>s = s + n</code>	13 <code>print(n, a)</code>	19 <code>print(n)</code>
7 <code>print(n, s)</code>		

1. Donner les affichages réalisés par ces scripts.
2. Déterminer le nombre de tours de chacune des boucles while.
3. Quels sont les scripts qui peuvent être écrits à l'aide d'une boucle bornée ?

Exercice 5 – (Sur papier puis à vérifier sur ordinateur)

On considère la fonction `is_prime` ci-dessous.

```

1 def is_prime(n):
2     d = 2
3     stop = False
4     while d < n and not stop :
5         if n % d == 0:
6             stop = True
7         d = d + 1
8     return stop

```

1. Combien de tours de la boucle while sont effectués lors de l'appel `is_prime(77)` et quelle valeur est renvoyée ?
2. Même question pour l'appel `is_prime(79)`.

Exercice 6 – (Sur papier puis à vérifier sur ordinateur)

En utilisant une boucle non bornée, écrire un script qui affiche la somme des carrés des 100 premiers entiers.

Exercice 7 – (Sur papier puis à vérifier sur ordinateur)

On note H_n la somme des inverse des n premiers entiers naturels :

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

1. Écrire une fonction `somme_inverses(n)` qui prend en argument un entier n et renvoie la valeur de H_n .
2. À l'aide d'un script, déterminer le plus petit entier n tel que $H_n > 10$.

Exercice 8

On considère la fonction `is_perfect` ci-dessous.

```
1 def is_perfect(n):
2     s = 0
3     d = 1
4     while d < n:
5         if n % d == 0:
6             s = s + d
7             d = d + 1
8     return s == n
```

1. Que renvoie l'appel `is_perfect(6)` ? L'appel `is_perfect(16)` ? L'appel `is_perfect(28)` ?
2. Quel est l'objectif du script ci-dessous ?

```
1 n = 1
2 keep_searching = True
3 while keep_searching :
4     n = n + 1
5     if n % 2 == 1 and is_perfect(n):
6         keep_searching = False
7 print(n)
```

Exercice 9

Expliquer pourquoi les scripts ci-dessous ne se terminent pas et proposer des corrections en conséquence.

1 <i># script 1</i>	8 <i># script 2</i>	15 <i># script 3</i>
2 n = 0	9 n = 0	16 x = 0
3 s = 0	10 s = 0	17 n = 0
4 while n < 10:	11 while s < 1000:	18 while x != 10:
5 s = s + n	12 n = n + 1	19 x = x + 0.1
6	13	20 n = n + 1
7 print(s)	14 print(n)	21 print(n)