La boucle non bornée

Définition

Une boucle non bornée consiste en une série d'instructions qui sont exécutées en boucle tant qu'une condition est vérifiée. Cette condition, dite de continuation, 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	4	-
2	4	0
3	2	0
4	2	0
5	1	0
6	1	1
7	0	1

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.

Une boucle bornée peut ne jamais s'arrêter (en théorie).

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	-	1
2	True	1
3	True	2
2	True	2
3	True	4
2	True	4
3	True	8
2	True	8
3	True	16
2	False	16

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.

```
# script 1
n = 5
s = 0
while n > 0:
    n = n - 1
s = s + n
print(n, s)
```

```
n = 0
a = 1
while n < 8:
    n = n + 1
a = a * 2
print(n, a)
```

```
n = 0
a = 1
while a < 1000:
    a = a * 2
n = n + 1
print(n)
```

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.

```
def is_prime(n):
    d = 2
    stop = False
    while d < n and not stop :
        if n % d == 0:
            stop = True
        d = d + 1
    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 :

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

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.

```
def is_perfect(n):
    s = 0
    d = 1
    while d < n:
        if n % d == 0:
            s = s + d
        d = d + 1
    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 ?

```
n = 1
keep_searching = True
while keep_searching :
    n = n + 1
    if n % 2 == 1 and is_perfect(n):
        keep_searching = False
print(n)
```

Exercice 9

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

```
# script 1
n = 0
s = 0
while n < 10:
    s = s + n

print(s)
```

```
# script 2
n = 0
s = 0
while s < 1000:
    n = n + 1

print(n)
```

```
# script 3
x = 0
n = 0
while x != 10:
    x = x + 0.1
    n = n + 1
print(n)
```