Type de document	Cours et exercices	Classe	1re	Durée	2h	Date	07/11/2022
Thème et contenu(s)	Langages et programmation – Boucles						
Capacités attendues	Bases de Python : notion d'itérable, utilisation dans les boucles bornées						
Prérequis	Comprendre le fonctionnement des boucles bornées						
Description	Introduction aux boucles non bornées et comparaison avec les boucles bornées						

I) Rappels

Nous avons jusqu'à présent travaillé sur des **boucles bornées**, que l'on programme en Python grâce au mot-clef `for`. Ces boucles se « terminent » : **on sait toujours à l'avance le nombre d'itérations qu'elles vont faire**. En effet, un itérable est un ensemble fini (on sait combien d'éléments il contient) : pour vous en convaincre, pensez à une chaîne de caractères. Même très grande, elle se possède une fin.

II) Boucles non-bornées

II - a) Définition et syntaxe

Une boucle **non bornée** est une boucle dont le nombre d'itérations n'est pas connu à l'avance. Un exemple extrême est la **boucle infinie**, qui peut être volontaire ou non.
Une boucle non bornée s'introduit avec le mot-clef `while` suivi d'une condition (exactement le même type de condition que vous pourriez mettre dans un `if`). Vous pouvez placer à la ligne autant d'instructions que vous voulez, le tout étant de bien penser à indenter (comme pour le `for`, le `if`). Tant que la condition n'est pas remplie, tout ce qui suit la boucle n'est pas exécuté et le programme « tourne » uniquement à l'intérieur de la boucle.

II - b) Exemple simple

Contrairement au `for`, avant d'utiliser une boucle non bornée (donc un `while`) on doit **initialiser une variable** qui nous servira de compteur (voir ci-contre). Cette variable doit être initialisée avant la boucle.

```
1  i = 100                    100
2  while i<105:               101
3      print(i)               102
4      i = i+1                103
5                             104
```

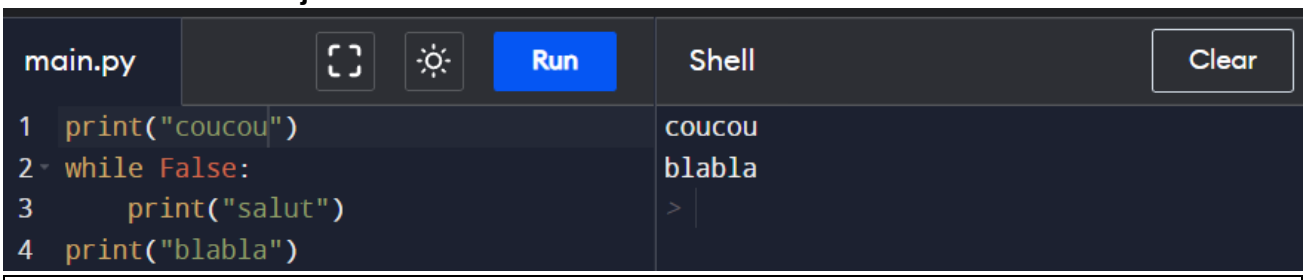
Si on ne fait pas **évoluer** cette variable (comme on l'a fait ci-dessus en lui ajoutant 1 à chaque itération) on tombe dans une **boucle infinie** c'est-à-dire une boucle de laquelle on ne sort jamais car sa condition est **toujours** évaluée à `True`. Dans le cas suivant, la valeur 100 sera affichée indéfiniment, et aucune instruction suivant cette boucle ne sera exécutée. En effet, `i` n'évolue pas, il est toujours inférieur à 105 donc l'expression logique `i<105` sera toujours évaluée à `True` par l'interpréteur Python. La ligne 3 est répétée indéfiniment.

```
1  i = 100                    100
2  while i<105:               100
3      print(i)               100
4                             100
5                             100
6                             100
7                             100
                              100
                              100
                              100
                              100
                              100
                              100
```

```
while True:
    print("je m'affiche indéfiniment")
```

```
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
je m'affiche indéfiniment
```

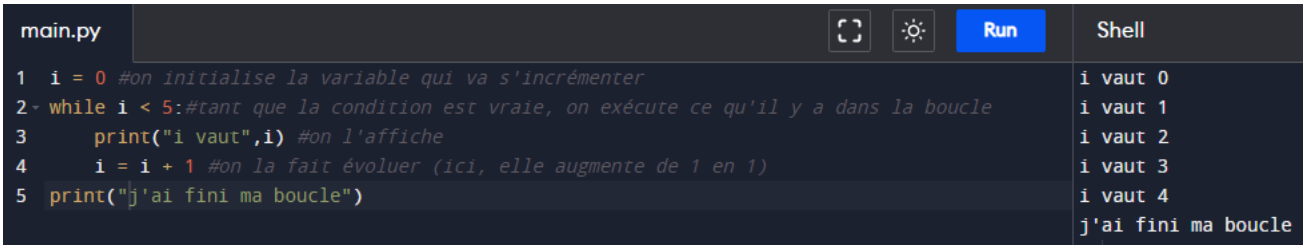
De la même manière, le code situé dans un « `while false` » ne sera **jamais** exécuté, le booléen `False` étant toujours évalué à … `False`. Dans la capture d'écran suivante, la chaine de caractères « salut » ne sera donc **jamais** affichée.



Remarque importante : attention à ne pas vous embrouiller, les `range()` ne sont utilisés **que** dans les boucles `for` ! Et **pas** dans les boucles `while` ! Pareil pour le mot-clef `in`

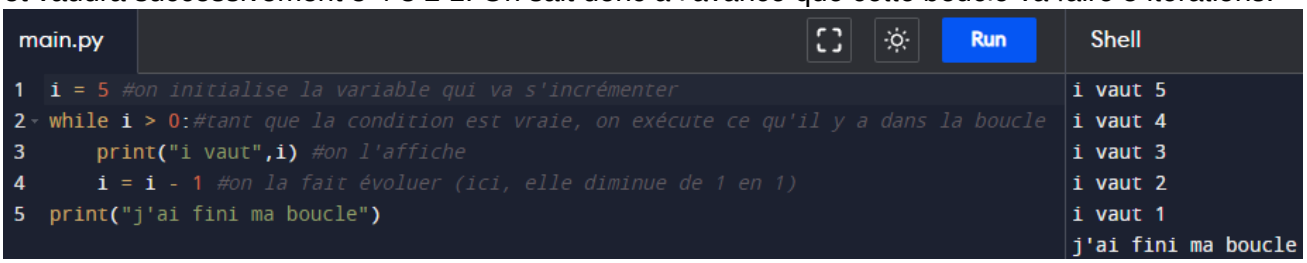
III) Exemple avec une variable qui s'incrémente

Dans l'exemple ci-dessous, on utilise une variable `i` initialisée à 0 qui va s'incrémenter à chaque « tour de boucle ». Cette boucle non-bornée se comporte comme une boucle bornée : on connaît à l'avance le nombre d'itérations qu'elle va faire (5 itérations, `i` valant successivement 0 1 2 3 4)



IV) Exemple avec une variable qui se décrémente

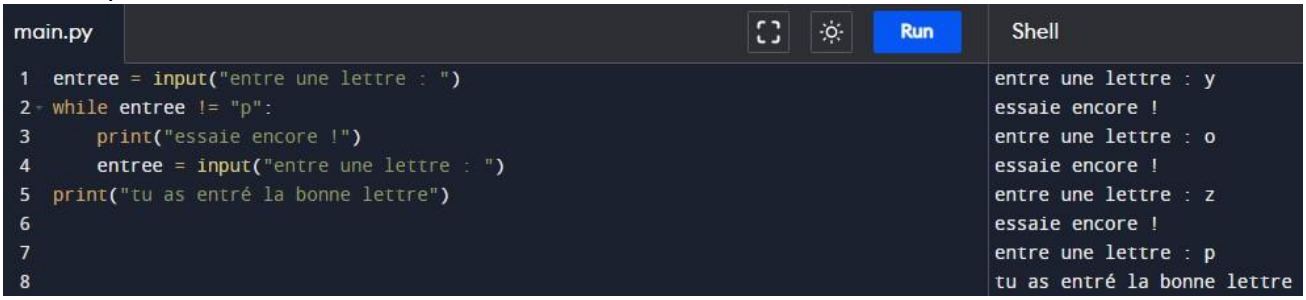
Même exemple qu'avant mais cette fois-ci avec `i` qui diminue de 1 en 1 à chaque « tour de boucle » et vaudra successivement 5 4 3 2 1. On sait donc à l'avance que cette boucle va faire 5 itérations.



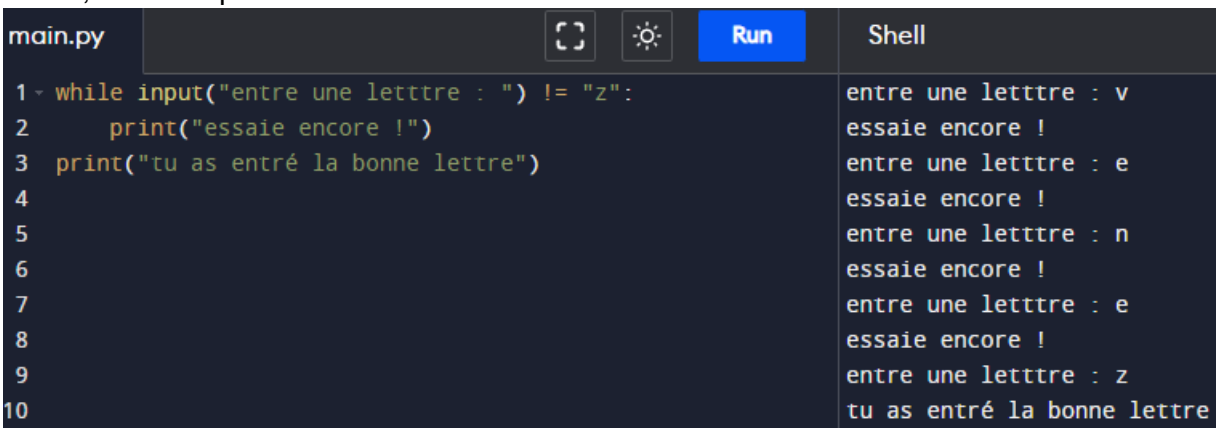
V) Exemple avec une condition dépendant de ce qu'entre l'utilisateur

Ici le nombre d'itérations n'est pas connu avant l'exécution du programme, il dépend de ce qu'a entré l'utilisateur au départ. On peut quand même remplacer cette boucle non-bornée `while` par une boucle bornée `for` car au début de l'exécution du programme c'est l'utilisateur qui fixe le nombre d'itérations qu'il veut faire (ci-dessous, ce nombre est 8).

En revanche, dans l'exemple suivant, le fait de sortir de cette boucle dépend de ce qu'entre l'utilisateur. On lui redemande à chaque fois d'entrer une lettre tant que cette lettre n'est pas égale à la lettre p. Le nombre d'itérations est donc totalement inconnu.



On peut même améliorer l'exemple précédent. En effet, la ligne 1 est nécessaire pour initialiser la variable `entree`. Mais on pourrait l'inclure dans la condition du `while`. Si vous n'êtes pas à l'aise avec ceci, n'utilisez pas cette méthode.



VI) Boucles imbriquées

On a souvent besoin, en informatique, d'imbriquer des boucles (bornées et bornées, non-bornées et non-bornées, ou un mélange des deux). Par exemple, on peut demander à l'utilisateur d'entrer un nombre et d'afficher chaque chiffre $c_1 \ldots c_n$ de ce nombre c fois. Par exemple si le nombre est 531, on doit afficher 5x le chiffre 5, 3x le chiffre 3, 1x le chiffre 1. Ici on va utiliser une boucle bornée pour parcourir la chaîne de caractères entrée par l'utilisateur élément par élément (c'est le plus simple) puis une boucle non bornée pour afficher chaque caractère le nombre de fois requis.

```
n = input("entre un nombre : ")
for e in n : #on parcourt la chaîne de caractères n élément par élément
    i = int(e) #initialisation de i : il vaut le "digit" courant
    while i>0:
        print(e)
        i = i - 1 #on décrémente i
```

```
entre un nombre : 531
5
5
5
5
5
3
3
3
1
```

Dans l'exemple suivant, on va parcourir tous les nombres entre 2^8 et 2^{12} et compter combien sont premiers. On le fait avec des boucles `while` mais on pourrait utiliser uniquement des `for`, ou bien un mélange des deux. Vous devez être capables de faire des exercices d'une difficulté égale à celle de celui-ci (difficulté facile-moyenne).

main.py Run

```
compteur = 0 #initialisation du compteur
i = 2**8 #on démarre de 2 puissance 8
while i <= 2**12: #tant que i est inférieur ou égal à 2 puissance 12...
    premier = True #on initialise une variable premier qui vaut True
    j=2 #on initialise j à 2, premier diviseur potentiel de i (après 1)
    while j<i: #tant que j est inférieur à i
        if i%j == 0: #si i est divisible par j...
            premier = False #...il n'est pas premier
        j = j + 1 #j évolue "quoi qu'il arrive" sinon...boucle infinie
    if premier == True: #si on a trouvé 0 diviseurs pour i...
        compteur = compteur + 1 #il est premier donc on le compte !
    i = i + 1
print("j'ai trouvé",compteur,"nombres premiers") #on affiche le résultat
```

Shell

```
j'ai trouvé 510 nombres premiers
>
```