2.2 Instructions de répétition

2.2.1 La boucle while *(Tant que)*

L'instruction de répétition appelée boucle permet d'exécuter plusieurs fois consécutives un même bloc d'instructions. La répétition s'effectue tant que la valeur de l'expression booléenne reste vraie (ou est égale à TRUE)

Remarque 5 :

Syntaxe de la boucle `while`

```
tantque (condition booléenne ):
        instructions
```

La `condition booléenne` indique la condition de sortie de cette boucle qui est, en fait, la négation de l'expression booléenne, c'est à dire `non(condition booléenne)`

Exemple 5 :

L'algorithme suivant affiche les 5 premiers entiers, en utilisant la boucle `while`

```
#-*-coding:Latin-1 -*

#algo affichage_des_premiers_entiers

compteur=1 #initialisation du compteur
while compteur<=5 :
  print(compteur)
  compteur=1+compteur
```

Commentaires :
La variable `compteur` est initialisée **avant** la boucle.

Exercice 3 :

Ecrire un algorithme qui affiche les n premiers entiers, n étant choisi par l'opérateur. (*Algorithme* `affichageDesNPremiersEntiers1`)

Remarque 6 :

1. Il y a plusieurs algorithmes qui peuvent être équivalents. Par exemple, la condition `(compteur<=5)` est équivalente `(compteur<6)` ; il y a donc plusieurs algorithmes possibles.
2. Pour écrire une boucle, il faut :
   (a) **Chercher la condition d'arrêt**
   (b) **Ecrire la négation de cette condition d'arrêt**, qui peut être tout aussi intéressante à utiliser.
3. Pour écrire une boucle, 3 étapes sont obligatoires :
   (a) Initialiser le compteur avant d'entrer dans la boucle
   (b) Etablir la condition de poursuite : il y a toujours des conditions de poursuite différentes, mais équivalentes
   (c) La modification d'une valeur dans la boucle *(celle que l'on a initialisée)*, pour que la répétition exprime une évolution des calculs

Remarque 7 :

Remarque importante
En Python, il y a une version originale de l'incrémentation qu'il est plus simple d'utiliser :

1. L'instruction

```
compteur = compteur +1
```

c'est à dire qu'on incrémente 1 à la variable `compteur` est équivalente à

```
compteur += 1
```

2. De la même manière, l'instruction

```
compteur = compteur *5
```

c'est à dire qu'on multiplie la variable `compteur` par 5 est équivalente à

```
compteur *= 5
```

Intéressant, non ?

2.2.2 Boucle et conditionnelle ; boucles imbriquées

On peut mélanger les différentes instructions de conditionelle et les boucles tout comme on peut faire des boucles imbriquées.

Exemple 6 :

1. Ce programme affiche le maximum de 5 nombres donnés par l'utilisateur

```
#-*-coding:Latin-1 -*
#algo MaxDe_5_Entiers

#Cet algorithme fait entrer 5 nombres en file et en affiche le plus grand des 5

valeur=float(input("Donner la première valeur: "))
compteur=1
max = valeur
while compteur<5 :
    valeur=float(input("Donner une autre valeur: "))
    if valeur>max :
                    max=valeur
    compteur+=1

print("Le plus grand des 5 nombres est",max);
```

2. L'algorithme suivant affiche la meilleure des 5 notes comprises entre 0 et 20 ; pour la programmation, attention à l'indentation ! !

```
#-*-coding:Latin-1 -*

#algo Note Max Entre 0 Et 20
#Cet algorithme donne la meilleure des 5 notes comprises entre 0 et 20



note= int(input("Donner une note entière entre 0 et 20 "))
while note<0 or note>20 :
  print("Vous avez fait une erreur")
```

```
    note= int(input("On recommence: donner une note entière entre 0 et 20 "))
maximum =note
compteur = 1
while compteur < 5 :
  note= int(input("Donner une note "))
  while note<0 or note>20 :
    print("Vous avez fait une erreur")
    note= int(input("Donner une note "))
  if note>=maximum :
    maximum=note
  compteur = 1 + compteur

print("La note maximale est ",maximum)
```

Exercice 4 :

Dans l'exemple précédent, nous avons l'instruction

```
while note<0 or note>20 :
```

Exprimer le test en utilisant la négation de cette condition