# TP 04 – Les boucles "While"

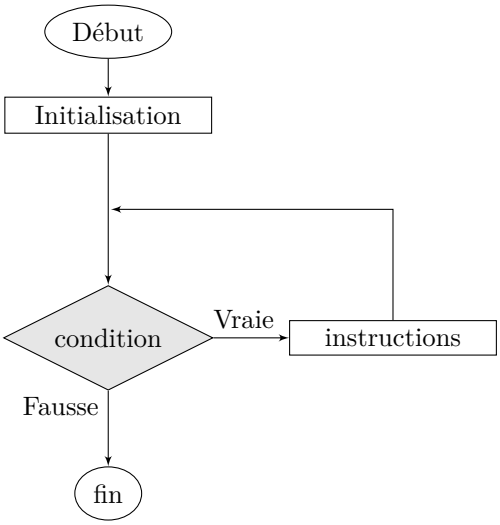
Pour répéter plusieurs fois un bloc d'instructions, on dispose en Python de deux structures de boucles : `for` et `while`. Essentiellement, on utilise une boucle `for` lorsque l'on sait combien de répétitions on doit faire, et une boucle `while` lorsque le nombre de répétitions nécessaires n'est pas connu à l'avance (dans ce cas, l'arrêt de la boucle est liée à une condition).

## I Principe d'une boucle while

La boucle `while` permet de réaliser un bloc d'instruction **tant qu'**une certaine condition est vraie.

- **Algorithme.**
  Initialisation
  *Tant que* la condition est vraie,
  Réaliser les instructions
  Fin
- **Syntaxe Python.**

```
initialisation
while condition:
    instructions
```




La condition de la boucle `while` doit être un *booléen*, c'est-à-dire un opérateur de test. Rappelons les principaux opérateurs de test.

| Égalité | == | Non-égalité | != |
|---|---|---|---|
| Inférieur strict | < | Supérieur strict | > |
| Inférieur ou égal | <= | Supérieur ou égal | >= |
| Appartenance | in | Non-Appartenance | not in |

Une boucle `for` est une boucle toujours finie (le nombre maximal de répétitions est égal au nombre d'éléments dans la séquence). Avec les boucles `while`, il y a un risque de créer une **boucle infinie**. En T.P., vous pourrez repérer les boucles infinies en remarquant que l'exécution de votre cellule est très longue : tant qu'il est indiqué `Entrée [*]`, l'exécution n'est pas terminée. Dans ce cas, vous aurez la possibilité d'interrompre l'exécution avec le bouton "stop" ■.

**Exercice 1** Donner l'affichage du programme suivant.

Entrée [1]:
```
n = 3
while n>= 0:
    print(n)
    n=n-1
```

| Valeurs de n | Condition vérifiée ? | Instructions réalisées |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

Out [1]:

▶ **Première erreur à ne pas faire :** Oublier d'**initialiser** la condition

Entrée [2]:
```
while n>= 0:
    print(n)
    n=n-1
```

Out [2]:
```
----> 1 while n >= 0:
      2     print(n)
      3     n=n-1

NameError: name 'n' is not defined
```

▶ **Deuxième erreur à ne pas faire :** Avoir une condition qui n'**évolue** pas

Entrée [3]:
```
n = 3
while n>= 0:
    print(n)
```

Out [3]:
```
3
3
3
... # boucle infinie
```

▶ **Troisième erreur à ne pas faire :** Avoir une condition qui ne devient jamais **fausse**

Entrée [4]:
```
n = 3
while n>= 0:
    print(n)
    n=n+1
```

Out [4]:
```
3
4
5
... # boucle infinie
```

**Exercice 2** Le programme suivant permet de déterminer, pour un entier $n$ donné, le plus petit entier $p$ tel que la condition $2^p \geqslant n$ est vérifiée. Pour cela, pour une variable `n` dont la valeur est fixée, on créé une boucle qui fait prendre à la variable `p` les valeurs successives des entiers tant que l'on est pas satisfait, c'est-à-dire tant que `2**p<n` (on s'arrête dès que cette condition n'est pas vérifiée, c'est-à-dire dès que `2**p>=n`.)

Entrée [5]:
```
n = 15
p = 1
while 2**p < n :
    p=p+1
print(p)
```

Donner l'affichage du programme suivant.

| Valeurs de `p` | Condition vérifiée ? | Instructions réalisées |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Out [5]:

## II Lecture de programme (sur papier)

**Exercice 3** Donner l'affichage des programmes suivants. Attention, certains programmes ne conduisent pas à un affichage de Python, mais à boucle infinie. Dans ce cas là, noter seulement que le programme est invalide et repérer la ligne du code qui pose problème.

Entrée [6]:
```
i = 0
while i <= 3 :
    print(i)
    i = i+1
```

Out [6]:

Entrée [7]:
```
i = 0
while i <= 3 :
    i = i+1
    print(i)
```

Out [7]:

Entrée [8]:
```
i = 0
while i <= 3 :
    i = i+1
print(i)
```

Out [8]:

Entrée [9]:
```
i = 0
while i <= 3 :
    i = i-1
print(i)
```

Out [9]:

Entrée [10]:
```
i = 0
while i != 3 :
    print(i)
    i = i+1
```

Out [10]:

Entrée [11]:
```
i = 0
while i != 3 :
    i = i+1
print(i)
```

Out [11]:

Entrée [12]:
```
i = 0
while i == 3 :
    i = i +1
print(i)
```

Out [12]:

Entrée [13]:
```
i = 3
while i == 3 :
    print(i)
```

Out [13]:

Entrée [14]:
```
N = 55
while N > 10:
    N = N - 10
print(N)
```

Out [14]:

Entrée [15]:
```
x = 1
while x<10:
    print('x a pour valeur', x)
    x = 2*x
print('au revoir')
```

Out [15]:

Pour le programme suivant, noter que la commande `break` sert à arrêter une boucle. Lorsque Python rencontre `break`, il arrête immédiatement la boucle (tout ce qui suit dans la boucle n'est pas exécuté) et passe à l'instruction suivante.

Entrée [16]:
```
n=5
while n>0 :
    print ("valeur de n : ", n)
    n=n-1
    if n%3==0 :
            print("on sort de la boucle")
            break
            print("au revoir")
print("fin du programme")
```

Out [16]:

## III Exercices (sur ordinateur) (que des suites...)

**Exercice 4** On considère un appartement coûte 100 000 euros cette année (que l'on considère comme l'année 0). Sa valeur augmente de 1% chaque année (c'est-à-dire que chaque année le prix de l'appartement est multiplié par 1.01.)

1. Que calcule le programme suivant ?

Entrée [17]:
```
prix = 100000
prix = prix*1.01
print(prix)
```

Out [17]:

2. Que calcule le programme suivant ?

Entrée [18]:
```
prix = 100000
prix = prix*1.01
prix = prix*1.01
print(prix)
```

Out [18]:

3. Ré-écrire le programme de la question précédente l'aide d'une boucle `for`.

Entrée [19]:
```
```

Out [19]:

4. Compléter le programme suivant qui permet, à l'aide d'une boucle `for`, de connaître la valeur de l'appartement au bout de 10 ans. ***Vérification*** *: On doit obtenir environ* `110462`.

Entrée [20]:
```
prix = 100000 #prix annee 0
for k in range(..... , ......):
    ......................... #reactualisation du prix chaque annee
print("Le prix au bout de 10 ans vaut ", prix)
```

Out [20]:

5. Compléter le programme suivant qui renvoie la liste de toutes les valeurs prises par l'appartement au cours de ses 10 ans. ***Vérification*** *: Le dernier terme de la liste obtenue doit valoir environ* `110462.21`.

Entrée [21]:
```
prix = 100000 #prix annee 0
L = [prix] #Creation d'une suite qui contient le prix de depart
for k in range(...... , ......):
    ......................... #reactualisation du prix chaque annee
    ......................... #On ajoute le nouveau prix à L
print(L)
```

6. Compléter la fonction suivante, appelée `prixappart`, qui prend en argument un entier `n` et qui renvoie le prix de l'appartement au bout de `n` années. ***Vérification*** *: L'évualuation de la fonction en* `n=10` *doit donner environ* `110462`.

Entrée [22]:
```
def prixappart(..........................):
    prix = 100000 #prix annee 0
    for k in range(1,..........................):
        ......................... #reactualisation du prix chaque annee
    return(prix)
```

Entrée [23]:
```
prixappart(10)
```

Out [23]:

7. Compléter le programme suivant qui permet de donner le nombre d'années au bout desquelles le prix de l'appartement aura doublé. ***Vérification*** *: On doit obtenir* `70`.

Entrée [24]:
```
prix = 100000
annee = 0
while prix < ......................... :
    ......................... #on actualise le prix
    ......................... #on actualise l'annee
print(annee)
```

Out [24]:

**Exercice 5** On considère un appartement coûte 100 000 euros cette année (que l'on considère comme l'année 0). Sa valeur augmente de 1234 euros chaque année. Répondre aux questions suivantes en s'inspirant de l'Exercice précédent.

1. Écrire un programme qui permet, à l'aide d'une boucle `for`, de connaître la valeur de l'appartement au bout de 10 ans. ***Vérification*** *: On doit obtenir* `112340`.

Entrée [25]:
```
```

Out [25]:

2. Définir une fonction Python, appelée `prixappart`, qui prend en argument un entier `n` et qui renvoie le prix de l'appartement au bout de `n` années. ***Vérification*** *: L'évualuation de la fonction en* `n=10` *doit donner environ* `1112340`.

Entrée [26]:
```
```

Entrée [27]:
```
prixappart(10)
```

Out [27]:

3. Écrire un programme qui permet de donner le nombre d'années au bout desquelles le prix de l'appartement aura dépassé 200 000 euros. ***Vérification*** *: On doit obtenir* `82`.

Entrée [28]:
```
```

Out [28]:

**Exercice 6** On considère la suite $(v_n)_{n\in\mathbb{N}}$ définie par, pour tout $n \in \mathbb{N}$, $v_n = \frac{1}{2n+1}$.

1. Écrire un programme, qui demande à l'utilisateur un entier `n` et qui renvoie la valeur le $n$-ième terme de la suite. ***Vérification*** *: Pour* `n=2`*, on doit obtenir* `0.2`.

Entrée [29]:
```
```

2. Écrire une fonction Python, qui prend en argument un entier `n` et qui renvoie la valeur du $n$-ième terme de la suite. ***Vérification*** *: On vérifiera que l'évaluation de cette fonction en* `2` *donne* `0.2`.

Entrée [30]:

3. Écrire un programme qui permet, à l'aide d'une boucle `for`, d'afficher les 20 premiers termes de la suite $(v_n)_{n\in\mathbb{N}}$ (de $v_0$ à $v_{19}$). ***Vérification*** *: Le dernier terme affiché doit valoir environ* `0.0256`.

Entrée [31]:

4. Écrire un programme, qui demande à l'utilisateur un entier `n` et qui renvoie la liste des $n+1$ premiers termes (de $u_0$ à $u_n$). ***Vérification*** *: Pour* `n=3`*, on doit obtenir la liste* `[1.0, 0.3333333333333333, 0.2]`.

Entrée [32]:

5. Écrire un programme permettant de déterminer le plus petit entier naturel $n$ tel que $v_n < 0,001$. ***Vérification*** *: On doit obtenir* `500`.

Entrée [33]:

**Exercice 7** EMLyon 2023, Maths E

Pour $x \in ]0,+\infty[$, on pose : $f(x) = \frac{e^{-x}}{x}$. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et par la relation de récurrence $u_{n+1} = f(u_n)$ valable pour tout entier naturel $n$.

1. Recopier et compléter la fonction Python suivante afin que l'appel `fonc_1(a)` renvoie le plus petit entier $n$ tel que $u_n > a$.

Entrée [34]:
```
def fonc_1(a):
    from numpy import exp
    u=1
    n=0
    while ..................... :
        u=exp(-u)/u
        n=..............
    return n
```

2. On considère maintenant la fonction `Python` :

```
def fonc_2(a):
    from numpy import exp
    u=1
    n=0
    while u>a:
        u=exp(-u)/u
        n=n+1
    return n
```

Les appels `fonc_1(10**6)` et `fonc_2(10**(-6))` donnent respectivement 6 et 5. Qu'en déduire pour $u_5$ et $u_6$ ? Commenter ce résultat en une ligne.

3. Écrire une fonction Python qui a pour argument un entier $n$ et qui renvoie la valeur de $u_n$. ***Vérification*** *: On vérifiera que l'évaluation de cette fonction en* `6` *renvoie environ* `1007430`.

Entrée [35]:

Entrée [36]:

Out [36]:

**Exercice 8** Soit $(v_n)_{n\in\mathbb{N}^*}$ une suite définie par

$$v_1 = 2 \qquad \text{et} \qquad \forall n \in \mathbb{N},\quad v_{n+1} = \exp\left(\frac{v_n}{n+1}\right)$$

1. Écrire un programme qui permet d'afficher les 20 premiers termes de la suite $(v_n)_{n\in\mathbb{N}}$ (de $v_1$ à $v_{20}$). Conjecturer la limite de $(v_n)$. ***Vérification*** *: Le dernier terme affiché doit valoir environ* `.05429`.

Entrée [37]:

2. Écrire un programme permettant de déterminer le premier $n \in \mathbb{N}$ tel que $v_n \leqslant 1.01$. ***Vérification*** *: On pourra vérifier que* $n = 102$.

Entrée [38]:

3. Écrire un programme permettant de déterminer le plus grand $n \in \mathbb{N}$ tel que $v_n \geqslant 1.005$. ***Vérification*** *: On pourra vérifier que* $n = 201$.

Entrée [39]:

# IV Exercices (sur l'ordinateur) (pas sur les suites)

**Exercice 9** Une énigme avec une feuille de papier et la Tour Eiffel. On dispose d'une feuille de papier d'épaisseur 0.1mm. On suppose qu'on peut la plier autour de fois que nécessaire. Combien de fois doit-on la plier au minimum pour que l'épaisseur dépasse la taille de la tour d'Eiffel (324m) ? Écrire un programme Python qui permet de résoudre ce problème. ***Vérification*** *: On doit trouver* `22`.

Entrée [40]:

Out [40]:

**Exercice 10** Afficher le temps qui passe... Écrire un programme qui affiche toutes les heures et minutes d'une journée. On doit donc obtenir "0 heure(s) 0 minute(s)" puis "0 heure(s) 1 minute(s)", "0 heure(s) 2 minute(s)",..., "0 heure(s) 59 minute(s)", "1 heure(s) 0 minute(s)",...

Entrée [41]:

**Exercice 11** On joue aux dés ! Dans cet exercice, on souhaite jouer au jeu suivant : on lance à chaque fois deux dés simultanément jusqu'à obtenir une paire (la même valeur pour les deux dés). Simuler cette expérience et compter le nombre de lancers nécessaire pour obtenir une paire. Afficher également la valeur de la paire (le nombre en commun sur les deux dés). *Pour le tirage d'un dé, on pourra charger la bibliothèque numpy.random grâce à l'instruction* `import numpy.random as rd` *et on pourra utilisera la commande* `rd.randint(a,b+1)` *qui renvoie un nombre entier aléatoire entre* `a` *et* `b`.

Entrée [42]:

**Exercice 12** Python nous aide à réviser nos multiplications ! Écrire un programme Python qui permet de vous entrainer sur vos tables de multiplication. Le programme commencera par vous demander un nombre maximal, puis tirera au sort deux nombres entre 2 et le nombre maximal choisi. Ensuite, il doit présenter la multiplication à effectuer, et vous demander votre réponse tant que celle-ci n'est pas la bonne. *Pour le tirage au sort d'un nombre, on pourra charger la bibliothèque numpy.random grâce à l'instruction* `import numpy.random as rd` *et on pourra utilisera la commande* `rd.randint(a,b+1)` *qui renvoie un nombre entier aléatoire entre* `a` *et* `b`.

```
#Exemple d'une partie
>> nombre maximum ? 15
>> 8x7
>> réponse ? 37
>> Faux ! Réessayer !
>> réponse ? 56
>> Bravo !
```

Entrée [43]:

**Exercice 13** Python nous aide à réviser nos multiplications (sous pression) ! Reprendre le programme précédent nous permettant de nous entrainer avec Python sur nos multiplications et l'améliorer de la manière suivante : à la fin de la partie, affichez le nombre d'erreurs et le temps utilisé pour répondre à la question. *On pourra utiliser la commande* `time.time()` *après l'importation* `import time` *qui donne le nombre de secondes écoulées depuis le 1er janvier 1970 à minuit pile.*

Entrée [44]:

**Exercice 14** Un peu d'arithmétique On cherche à écrire un programme qui, à un entier `n` donné, renvoie le plus grand l'entier `p` tel que $n = 2^p \times k$ avec $k$ un entier non divisible par 2 (c'est-à-dire un entier impaire donc...). Pour cela, on peut diviser par deux l'entier `n` tant que c'est possible et le nombre de divisions effectués donne l'entier `p` (cela revient à «supprimer tous les facteurs de 2 de l'entier `n`»). *On pourra utiliser les commandes* `a//b` *et* `a%b` *qui donnent respectivement le quotient et le reste de la division euclidienne de* `a` *par* `b`. ***Vérification*** *: Pour* `n=56`*, on doit trouver* `p=3` *car* $56 = 2^3 \times 7$.

Entrée [45]:

**Exercice 15** Fermat avait tord... Pierre de Fermat ($\sim$ 1605-1665) pensait que tous les entiers de la forme $F_n = 2^{2^n} + 1$ (avec $n$ un entier) étaient des nombres premiers. Effectivement, $F_0 = 3$, $F_1 = 5$ et $F_2 = 17$ sont des nombres premiers. S'il avait connu Python, il aurait sûrement changer d'avis ! Trouver le plus petit entier $n$ tel que $F_n$ n'est pas premier. *On pourra commencer par créer une fonction, permettant de tester si un nombre est premier ou non. Bon à savoir : pour tester si $n$ est premier, il suffit de tester les diviseurs jusqu'à $\sqrt{n}$, ce qui permet «d'économiser» des opérations.* ***Vérification*** *: On doit obtenir* `n=5`.

Entrée [46]:

**Exercice 16** Décomposition d'une fraction. On admet que tout rationnel $r = \frac{p}{q}$ avec $p$ et $q$ des entiers tels que $0 < p < q$ peut s'écrire sous la forme :

$$\frac{p}{q} = \frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n},$$

où $a_1, a_2, \ldots, a_n$ sont des entiers tels que $0 < a_1 < a_2 < \cdots < a_n$. L'objectif de cet exercice est d'écrire un programme permettant de calculer $a_1, a_2, \ldots, a_n$, étant donnés $p$ et $q$.

1. Ecrire une fonction prenant en arguments deux entiers naturels $a$ et $b$ et renvoyant le plus petit entier $n$ tel que $\frac{1}{n} \leqslant \frac{a}{b}$.

Entrée [47]:

2. Ecrire une fonction prenant en arguments quatre entiers $a, b, c, d$, et renvoyant deux entiers $e$ et $f$ tels que $\frac{e}{f} = \frac{a}{b} - \frac{c}{d}$.

Entrée [48]:

3. Ecrire une fonction prenant en arguments $p$ et $q$ et renvoyant une liste contenant les entiers $a_1, a_2, \ldots, a_n$.

Entrée [49]: