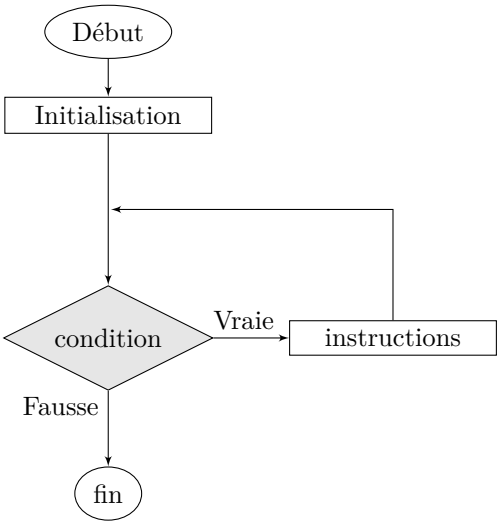
ALGO 06 – LES BOUCLES "WHILE"

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" ■.

II Étude de quelques programmes

II.1 Un premier programme et les erreurs classiques

Le programme suivant affiche un compte à rebours de 3 à 0.

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

Valeurs de n	Condition vérifiée ?	Instructions réalisées
3	Oui	Affiche 3 La variable n prend la valeur 2
2	Oui	Affiche 2 La variable n prend la valeur 1
1	Oui	Affiche 1 La variable n prend la valeur 0
0	Oui	Affiche 0 La variable n prend la valeur -1
-1	Non	Le programme s'arrête

Out [1]:
```
3
2
1
0
```

► **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
```

II.2 Deuxième programme

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)
```

Valeurs de `p`	Condition vérifiée ?	Instructions réalisées
1	Oui (car $2^1 = 2 < 15$)	La variable `p` prend la valeur 2
2	Oui (car $2^2 = 4 < 15$)	La variable `p` prend la valeur 3
3	Oui (car $2^3 = 8 < 15$)	La variable `p` prend la valeur 4
4	Non (car $2^4 = 16 \geqslant 16$)	Le programme s'arrête et renvoie la valeur 4

Out [5]:
```
4
```

III Troisième programme

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = \frac{e^{-u_n}}{u_n}$. À l'aide des programmes suivants, quelle conjecture peut-on faire sur la convergence de la suite $(u_n)_{n\in\mathbb{N}}$?

Entrée [6]:
```
import numpy as np
```

Entrée [7]:
```
u = 1
n = 0
while u > 0.00001:
    u = np.exp(-u)/u
    n = n + 1
print(n)
```

Out [7]:
```
5
```

Entrée [8]:
```
u = 1
n = 0
while u < 100000:
    u = np.exp(-u)/u
    n = n + 1
print(n)
```

Out [8]:
```
6
```

Détortiquons le premier programme.

u	n	u_n	Condition	Déduction
1	0	u_0	Oui	$u_0 > 0.0001$
	1	u_1	Oui	$u_1 > 0.0001$
....	....	...	...	...
	4	u_4	Oui	$u_4 > 0.0001$
	5	u_5	Non	$u_5 \leqslant 0.0001$

De l'étude des deux programmes, on en déduit que

$$u_5 \leqslant 0.0001 \qquad \text{et} \qquad u_6 \geqslant 10000$$

La suite $(u_n)_{n\in\mathbb{N}}$ semble donc osciller et ainsi ne pas admettre de limite.

IV Lecture de programme

Exercice 1 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 [9]:
```
i = 0
while i <= 3 :
    print(i)
    i = i+1
```

Out [9]:
```
0
1
2
3
```

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

Out [10]:
```
1
2
3
4
```

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

Out [11]:
```
4
```

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

Out [12]:
```
#Boucle infinie
```

Entrée [13]:
```
i = 0
while i != 3 :
    print(i)
    i = i+1
```

Out [13]:
```
0
1
2
```

Entrée [14]:
```
i = 0
while i != 3 :
    i = i+1
print(i)
```

Out [14]:
```
3
```

Entrée [15]:
```
i = 0
while i == 3 :
    i = i +1
print(i)
```

Out [15]:
```
0
```

Entrée [16]:
```
i = 3
while i == 3 :
    print(i)
```

Out [16]:
```
#Boucle infinie
```

Entrée [17]:
```
N = 55
while N > 10:
    N = N - 10
print(N)
```

Out [17]:
```
5
```

Exercice 2 On considère un appartement coûte 100 000 euros cette année. 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. 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.

Entrée [18]:
```
prix = 100000 #prix annee 0
for k in range(1,11):
    prix = prix*1.01 #reactualisation du prix chaque annee
print("Le prix au bout de 10 ans vaut ", prix)
```

Out [18]:
```
Le prix au bout de 10 ans vaut  110462.21254112048
```

2. Définir en Python en fonction, appelée `prixappart`, qui prend en argument un entier `n` et qui renvoie le prix de l'appartement au bout de `n` années.

Entrée [19]:
```
def prixappart(n):
    prix = 100000 #prix annee 0
    for k in range(1,n+1):
        prix = prix*1.01 #reactualisation du prix chaque annee
    return(prix)
```

Entrée [20]:
```
prixappart(10)
```

Out [20]:
```
110462.21254112048
```

3. Compléter le programme suivant qui permet de donner le nombre d'années au bout desquelles le prix de l'appartement aura doublé.

Entrée [21]:
```
prix = 100000
annee = 0
while prix < 2*100000:
    prix = prix*1.01
    annee = annee + 1
print(annee)
```

Out [21]:
```
70
```

Exercice 3 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.

Entrée [22]:
```
epaisseur = 0.0001
compteur = 0
while epaisseur<324:
    epaisseur = epaisseur*2
    compteur = compteur +1
print(compteur)
```

Out [22]:
```
22
```

Exercice 4 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 permet d'afficher les 20 premiers termes de la suite $(v_n)_{n\in\mathbb{N}}$ (de v_0 à v_{19}).

Entrée [23]:
```
for k in range(0,20):
    print(1/(2*k+1))
```

2. Écrire un programme permettant de déterminer le plus petit entier naturel n tel que $v_n < 0,001$. *On pourra vérifier que* $n = 500$.

Entrée [24]:
```
n = 0
v = 1
while v >= 0.001:
    n = n + 1
    v = 1/(2*n+1)
print(n)
```

Exercice 5 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).

Entrée [25]:
```
import numpy as np
v = 2
print(v)
for n in range(1, 20):
    v = np.exp(v/(n+1))
    print(v)
```

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

Entrée [26]:
```
n = 1
v = 2
while v > 1.01:
    v = np.exp(v/(n+1))
    n = n+1
print(n)
```

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

Entrée [27]:
```
n = 1
v = 2
while v >= 1.005:
    v = np.exp(v/(n+1))
    n = n+1
print(n-1)
```