

## Objectifs du TP :

- Savoir écrire un test
- Savoir écrire une boucle while
  - Choisir la condition
  - Incrémenter le variant de boucle.
- Savoir écrire une boucle for.
  - Bien choisir les valeurs du variant de boucle.
  - Savoir calculer une somme (comme dans l'exercice 10).
  - Savoir calculer les termes d'une suite récurrente (comme dans l'exercice 12).

## 1 Le branchement conditionnel

### Exercice 1. Majorité

On s'intéresse à la fonction suivante.

```
1 def majeur_mineur(age):
2     if age >= 18:
3         rep = 'majeur'
4     else:
5         rep = 'mineur'
6     return rep
```

1. Quelles sont les variables d'entrée et de sortie? La variable d'entrée est `age` et celle de sortie est `rep`.
2. De quel type sont-elles?  
La variable d'entrée est un nombre (type `int` ou `float` selon la valeur saisie).  
La variable de sortie est une chaîne de caractères (type `str`).

### Exercice 2. Valeur absolue

Écrire une fonction `valeur_absolue` qui reçoit en entrée un réel  $x$  et renvoie sa valeur absolue (sans utiliser la commande `abs`).

```
1 def valeur_absolue(x):
2     if x >= 0:
3         return x
4     else:
5         return -x
```

```
1 def valeur_absolue(x):
2     if x < 0:
3         x = -x
4     return -x
```

```
>>> valeur_absolue(2.5)
2.5
>>> valeur_absolue(-2.5)
2.5
```

### Exercice 3. Maximum de deux réels

Écrire une fonction `max2` qui calcule le maximum de deux nombres  $a$  et  $b$  qu'elle reçoit en entrée.

```
1 def max2(a,b):
2     if a >=b:
3         return a
4     else:
5         return b
```

```
1 def max2(a,b):
2     if a >=b:
3         return a
4     return b
```

```
>>> max2(12,-5)
12
```

### Exercice 4. Maximum de trois réels

En utilisant la fonction `max2`, écrire une fonction `max3` qui calcule le maximum de trois nombres  $a$ ,  $b$  et  $c$ .

```
1 def max3(a,b,c):
2     d = max2(a,b)
3     e = max2(d,d)
4     return e
```

```
1 def max3(a,b,c):
2     return max2(max2(a,b),c)
```

```
>>> max3(12,-5,15)
15
```

### Exercice 5. Médiane de trois nombres

Écrire une fonction `mediane` qui, lorsqu'on lui donne trois valeurs  $a$ ,  $b$ ,  $c$ , renvoie le nombre qui est compris entre les deux autres (on suppose que les trois nombres sont distincts deux à deux).

```
1 def mediane(a,b,c):
2     if a>b>c or c>b>a:
3         return b
4     if b>a>c or c>a>b:
5         return a
6     return c
```

```
>>> mediane(2,5,3)
3
```

## 2 Les boucles « tant que » et « pour »

### 2.1 La boucle « tant que »

#### Exercice 6. Puissance

On s'intéresse à la fonction suivante.

```
1 def puissance_de_3(n):
2     p = 0
3     while 3**p < n:
4         p = p + 1
5     return 3**p
```

1. Quelles sont les variables d'entrée et de sortie et quel est leur type?  
La variable d'entrée est un entier  $n$ .  
La variable de sortie est un entier,  $3^p$ .
2. Que renvoie la fonction si on l'exécute pour  $n=10$  ?
  - A l'initialisation,  $p = 0$ .
  - $3^0 < 10$  donc  $p = 1$
  - $3^1 < 10$  donc  $p = 2$
  - $3^2 < 10$  donc  $p = 3$
  - $3^3 > 10$  donc le programme s'arrête et la fonction renvoie  $3^2 = 9$ .
3. A quoi sert la ligne `p=p+1`?  
La ligne `p=p+1` correspond à l'incrément de la variable  $p$ . Elle augmente de 1 à chaque passage dans la boucle.

### Exercice 7. Partie entière

Écrire une fonction `partie_entiere` qui prend en entrée un réel  $x$  strictement positif et qui renvoie le plus grand entier inférieur à  $x$ .

```
1 def partie_entiere(x):
2     n=0
3     while n<x:
4         n=n+1
5     return n
```

```
>>> partie_entiere(12)
12
>>> partie_entiere(12.3)
13
>>> partie_entiere(-12.3)
0
```

## 2.2 La boucle « pour »

### Exercice 8. Découverte du range

On s'intéresse au programme suivant.

```
1 x=1
2 for i in range(5):
3     x=x+1
4     print('i=', i, 'et x = ',x)
5 print('la boucle est finie')
```

1. Que représente la commande `range(5)` ?  
La commande `range(5)` indique qu'on va exécuter 5 fois la tâche présente dans la boucle.
2. Que va-t-on obtenir dans le shell si on exécute ce programme ?

```
>>> (executing lines 40 to 44 of "tp2.py")
2
3
4
5
6
la boucle est finie
```

### Exercice 9. Somme

On s'intéresse à la fonction suivante :

```
1 def somme(n):
2     s = 0
3     for i in range(n):
4         s = s + i
5     return s
```

Que renvoie la fonction si on tape dans le shell `somme(4)` ?

Le tableau suivant permet de suivre l'évolution des variables.

valeur de $i$	valeur de $s$
	$s = 0$
$i = 0$	$s = 0$
$i = 1$	$s = s + 1 = 1$
$i = 2$	$s = s + 2 = 3$
$i = 3$	$s = s + 3 = 6$

La boucle s'arrête et la fonction renvoie 6.

### Exercice 10. Sommation

1. Écrire une fonction `somme2` qui prend en entrée deux entiers  $n$  et  $p$  et qui calcule la somme des entiers compris (au sens large) entre  $n$  et  $p$ .

```
1 def somme(n,p):
2     """
3     entrée : 2 entiers
4     sortie : un entier
5     But : calculer la somme des entiers de n à p
6     """
7     S=0
8     for i in range(n,p+1):
9         S=S+i
10    return S
```

2. Tester la fonction avec `somme2(3,8)` qui doit renvoyer 33.  
Le programme renvoie bien la valeur 33.

```
>>> somme2(3,8)
33
```

3. Écrire une fonction `somme_carres` qui prend en entrée deux entiers  $n$  et  $p$  et qui calcule la somme des carrés des entiers compris (au sens large) entre  $n$  et  $p$ .

```
1 def somme_carres(n,p):
2     """
3     entrée : 2 entiers
4     sortie : un entier
5     But : calculer la somme des carrés des entiers de n à p
6     """
7     S=0
8     for i in range(n,p+1):
9         S=S+i**2
10    return S
```

4. Tester la fonction avec `somme_carres(3,8)` qui doit renvoyer 199.  
Le programme renvoie bien la valeur 199.

```
>>> somme_carres(3,8)
199
```

### Exercice 11. Factorielle

1. Écrire une fonction `factorielle` qui prend en entrée un entier  $n$  et qui calcule  $n!$  (sans utiliser la fonction `factorial`).

```
1 def factorielle(n):
2     res=1
3     for i in range(1,n+1):
4         res=res*i
5     return res
```

2. Tester la fonction avec `factorielle(5)` qui doit renvoyer 120.  
Le programme renvoie bien la valeur 120.

```
>>> factorielle(5)
120
```

## 2.3 Application à l'étude des suites récurrentes

### Exercice 12. Suite définie par récurrence

On s'intéresse à la suite  $(u_n)_{n \in \mathbb{N}}$  définie par  $u_0 = 1$  et  $\forall n \in \mathbb{N}, u_{n+1} = 2u_n + 3$ .

1. Écrire une fonction `suite1` qui prend en entrée `n` et renvoie la valeur de  $u_n$ .

```
1 def suite1(n):
2     y=1 #initialisation
3     for i in range(1,n+1):
4         y=2*y+3
5     return y
```

2. Vérifier que le résultat obtenu pour `n=4` est 61.

On calcule  $u_4$  à la main :  $u_1 = 5$ ,  $u_2 = 2 * 5 + 3 = 13$ ,  $u_3 = 2 * 13 + 3 = 29$  et  $u_4 = 2 * 29 + 3 = 61$ .  
Le programme renvoie la même valeur.

```
>>> suite(4)
61
```

3. La suite  $(u_n)_{n \in \mathbb{N}^*}$  est maintenant définie par  $u_1 = 1$  et  $\forall n \in \mathbb{N}^*, u_{n+1} = 2u_n + 3$ .  
Écrire la fonction `suite2(n)` qui renvoie la valeur de  $u_n$ .

```
1 def suite2(n):
2     y=1 #initialisation
3     for i in range(2,n+1):
4         y=2*y-3
5     return y
```

### Exercice 13. Récurrence double

On s'intéresse à la suite  $(u_n)_{n \geq 5}$  définie par  $u_5 = 1$ ,  $u_6 = 2$  et  $\forall n \geq 5, u_{n+2} = u_n * u_{n+1}$ .

1. Écrire une fonction `suite3` qui prend en entrée `n` et renvoie la valeur de  $u_n$ .

```
1 def suite_bis(n):
2     a=1
3     b=2
4     for i in range(7,n+1): # i correspond à l'indice du nouveau terme calculé
5         a,b=b,a*b
6     return b
```

2. Vérifier le résultat obtenu pour `n = 10`.

On calcule  $u_{10}$  à la main :  $u_7 = u_5 * u_6 = 2$ ,  $u_8 = u_6 * u_7 = 4$ ,  $u_9 = u_7 * u_8 = 8$  et  $u_{10} = u_8 * u_9 = 32$ .  
Le programme renvoie la même valeur.

```
>>> suite_bis(10)
32
```

### Exercice 14. Suite convergente

On s'intéresse à la suite  $(u_n)_{n \in \mathbb{N}^*}$  définie par  $u_0 = 2$  et  $\forall n \in \mathbb{N}, u_{n+1} = \frac{1}{2} \left( u_n + \frac{2}{u_n} \right)$ .

1. Écrire une fonction `appro` qui prend en entrée une paramètre  $\epsilon$  et qui renvoie le premier entier `n` tel que  $|u_{n+1} - u_n| < \epsilon$ .

```
1 def appro(eps):
2     a=2
3     b=1/2*(a+2/a)
4     n=1
5     while abs(a-b)>eps:
6         a,b=b,1/2*(b+2/b)
```

```
7         n=n+1
8     return n
```

2. Déterminer une valeur approchée de la limite de cette suite à  $10^{-3}$  près puis à  $10^{-10}$  près.  
On modifie la fonction pour qu'elle rende également la valeur du terme de la suite correspondant au rang obtenu.

```
1 def appro_bis(eps):
2     a=2
3     b=1/2*(a+2/a)
4     n=1
5     while abs(a-b)>eps:
6         a,b=b,1/2*(b+2/b)
7         n=n+1
8     return n,b
```

## 3 Pour aller plus loin

### Exercice 15. Algorithme de Syracuse :

Voici les étapes de l'algorithme proposé par Syracuse :

- choisir un entier positif non nul (la **graine**),
- s'il est pair on le divise par 2, sinon on le multiplie par 3 et on ajoute 1,
- répéter le processus.

On constate que quelle que soit la graine choisie, la suite des valeurs passe par la valeur 1 (puis 4, 2, 1, 4, 2, 1, ...). Personne ne sait démontrer que cette affirmation est vraie !

On appelle **longueur de la chaîne** le nombre d'étapes nécessaires pour arriver à la première valeur 1.

1. Compléter, à la main, la suite des valeurs pour la graine 7 : 7, 22, 11, 34, 17, 52...  
On obtient 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
2. Quelle est la longueur de la chaîne pour la graine 7 ?  
On obtient une chaîne de longueur 16.
3. Écrire une fonction qui prend en entrée la graine, qui affiche la suite des nombres jusqu'à 1 et renvoie la longueur de la chaîne. On pourra utiliser la commande `print` pour afficher la suite des nombres.

```
1 def syracuse(graine):
2     a=graine #initialisation
3     cpt=0 # un compteur
4     while a!=1:
5         print(a)
6         cpt=cpt+1
7         if a%2==0:
8             a=a//2 # opération de division entre entiers
9         else:
10            a=3*a+1
11    return cpt
```

```
>>> syracuse(7)
7
22
11
34
17
52
26
13
```

40  
20  
10  
5  
16  
8  
4  
2  
16