

# 1 Fonctions

**Définition.** En informatique comme en mathématique, une *fonction* prend des valeurs en entrée, et renvoie (ou retourne) une nouvelle valeur.

**Exemple 1.** On considère la fonction suivante.

```
def mafonction(a, b):  
    return (a+b)/2
```

1. Que renvoient les appels `mafonction(14, 16)` et `mafonction(12, 20)` ?
2. À quoi sert cette fonction ?
3. Modifier cette fonction pour qu'elle calcule et renvoie la moyenne de trois nombres. Quelle instruction utiliser alors pour calculer la moyenne des trois nombres : 7, 19, 2 ?

# 2 Affectation

**Définition.** Une *variable* est un espace en mémoire, qui porte un nom, et à laquelle on peut affecter une valeur.

En Python, l'affectation se note `NOM = VALEUR`.

**Exemple 2.**

1. Que renvoie `tagada(0, 2)` ?
2. Que renvoie `tagada(3, 5)` ?

```
def tagada(a, b):  
    a = a + b  
    a = 2 * a  
    b = b - a  
    return b
```

**Exemple 3.**

1. Que renvoie `mystere(10)` ? Que renvoie `mystere(2)` ?  
Que remarquez-vous ?
2. Prouver votre conjecture.

```
def mystere(n):  
    n = n + 3  
    n = 6 * n  
    n = n - 18  
    n = n / 6  
    return n
```

**Exemple 4.** On souhaite écrire un algorithme qui, étant donné deux variables  $x$  et  $y$ , inverse leur contenu (c'est-à-dire qu'à la fin de l'exécution, la valeur de départ de  $x$  se trouve dans  $y$ , et la valeur de départ de  $y$  se trouve dans  $x$ ).

1. Exécuter l'algorithme avec différentes valeurs de  $x$  et  $y$ . Fonctionne-t-il ?
2. Corriger cet algorithme.

```
x = y
y = x
```

**Exemple 5.**

1. Exécuter l'algorithme avec différentes valeurs de  $x$  et  $y$ .
2. À quoi sert-il ?

```
x = x - y
y = x + y
x = y - x
```

### 3 Conditionnelles

**Définition.** Une *instruction conditionnelle* permet d'exécuter une partie du code uniquement si une *condition* est vérifiée.

**Exemple 6.**

1. Recopier et compléter le programme suivant pour que les fonctions `maximum` et `minimum` calculent et renvoient le plus grand (ou le plus petit) des nombres donnés en argument. Par exemple :

Appel	Résultat
<code>maximum(2, 7)</code>	7
<code>minimum(2, 7)</code>	2
<code>maximum(10, -1)</code>	10
<code>minimum(10, -1)</code>	-1

```
def maximum(a, b):
    ...

def minimum(a, b):
    ...
```

2. Quelle instruction utiliser pour calculer le maximum des deux nombres  $x$  et  $y$  ?

**Exemple 7.** Un salon de coiffure pratique les réductions suivantes :

- adulte : pas de réduction
- adolescent : 20% de réduction
- enfant : 50% de réduction

Compléter la fonction suivante pour qu'étant donné le prix et le statut du client, elle renvoie le prix après réduction.

```
def réduction(prix, statut):
    if statut == "enfant":
        prix = ...
    if statut == "adolescent":
        prix = ...
    return prix
```

**Exercice.** Activité 2 page 52.

## 4 Boucles bornées

**Propriété.** En Python, `range(n)` permet d'énumérer  $n$  nombres, de 0 à  $n-1$ .

**Exemple 8.**

- `range(3)` : .....
- `range(10)` : .....

**Définition.** Une *boucle bornée* permet de répéter des instructions un nombre de fois connu à l'avance.

**Exemple 9.** La fonction ci-contre calcule la somme des nombres entiers de 1 à  $n$ .

1. Pourquoi utiliser `range(n+1)` plutôt que `range(n)` ?
2. Que renvoie `somme(10)` ?
3. Modifier la fonction pour qu'elle calcule et renvoie la somme des carrés des entiers de 1 à  $n$ .

```
def somme(n):
    somme = 0
    for i in range(n+1):
        somme = somme + i
    return somme
```

**Exemple 10.** Dans un parc naturel, une population de marmottes augmente de 3% chaque année. Au début de l'étude, il y a environ 238 marmottes dans le parc.

1. Compléter : Augmenter une valeur de 3% revient à la multiplier par \_\_\_\_\_.
2. Compléter la fonction suivante pour que l'appel `marmottes(10)` calcule et renvoie le nombre de marmottes dix ans après le début de l'étude.

```
def marmottes(n):
    population = ...
    for i in ...
        ...
    return ...
```

**Exemple 11.** On simule à l'aide d'un programme informatique le lancer de pièces de monnaie, en comptant le nombre de « pile » obtenues.

Pour cela :

- on utilise la fonction `random()`, qui renvoie aléatoirement un nombre entre 0 et 1 ;
- on introduit une variable `compteur`, qui compte le nombre de « pile » obtenues jusqu'à présent.

1. Compléter la ligne 6 pour que la condition soit vraie une fois sur deux.
2. Compléter les autres lignes pour que la fonction compte le nombre de « pile » obtenues sur  $n$  lancers.

```

from random import random

def piles(n):
    compteur = ...
    for i in range(n):
        if random() > ...:
            compteur = ...
    return compteur

```

## 5 Boucles non bornées

**Définition.** Une *boucle non bornée* permet de répéter une instruction tant qu'une condition est vraie.

**Exemple 12.** On cherche le plus petit nombre entier dont le carré est supérieur à 1000.

1. Compléter la fonction ci-contre pour qu'elle renvoie ce nombre.
2. Quel est ce nombre ?

```

def carré():
    while ...:
        ...
    return ...

```

**Exemple 13.** Une population de bactéries est composée de 7 individus. Chaque heure, la population double.

1. Quelle sera la population au bout d'une heure ? De deux heures ?

On considère la fonction ci-contre, qui modélise l'évolution de la population de bactéries.

```

def bactéries():
    population = 7
    heure = 0
    while population < 1000:
        heure = heure + 1
        population = ...
    return heure

```

2. Exécuter l'algorithme, en notant les valeurs intermédiaires dans le tableau ci-dessous. Quelle est la valeur renvoyée par la fonction ?

Itération		1	2	3	4	5	6	7	8	9	10
Heure	0										
Population	7										

3. Au bout de combien d'heures la population sera-t-elle supérieure à 1000 bactéries ?