Corrigé – Les fonctions

Exercice 1

Fonctions :

- `addition` :
  - Paramètres : `a` et `b`
  - Variables locales : `somme`
  - Appels : aucun !
- `multiplication` :
  - Paramètres : `a` et `b`
  - Variables locales : `produit`
  - Appels : lignes 12 avec les paramètres 8 et 7 et ligne 14 avec les paramètres 3 et 6.

Un seul affichage est effectué ligne 15 : `5 56 3 18`

Exercice 2

De manière générale, les fonctions doivent être déclarées ensemble en haut du fichier, et le script à la fin.

```
1       a = 10 # déclarations de fonctions à faire en premier
2       def ma-fonction (x , y): # pas de tiret dans un nom de fonction et y ne sert pas
3           x = 0 # écrasement de paramètre
4           return a + x # a n'est pas une variable locale de la fonction
5
6       ma-fonction(8) # à mettre dans le script, après les définitions de fonctions,
        ↪  prend 2 paramètres
7       y = 5 # idem, dans le script
8       def Hello(): # pas de majuscule dans un nom de fonction, surtout au début
9           print(x + y) # ces variables n'existent pas dans la fonction
```

C'est rare de devoir faire un `print` dans une fonction, à moins que ce soit son but précis. Les `print` dans les fonctions serviront plutôt à débugger la fonction.

Exercice 3

1.
```
def fois_deux(x):
    produit = x*2
    print(produit)
```

2.
```
fois_deux(6) # affiche 12
```

3.
```
def fois_deux(x):
    produit = x*2
    return produit
```

4.
```
resultat = fois_deux(6) # n'affiche pas 12
print(resultat) # affiche 12
```

5.
```
def multiplication():
    produit = a * b
    print(produit)
```

6.
```
multiplication(3, 4) # affiche 12, en entier
```

7.
```
multiplication(3, 4) # affiche 12.0, un flottant
```

La même fonction renvoie des résultats de types différents en fonction des paramètre passés.

8.
```
def multiplication(a, b):
    produit = a * b
    return produit
```

9.
```
resultat = multiplication(3.0, 4.0) # n'affiche rien
print(resultat) # affiche le résultat
```

Faire le même test que dans la question 6, et récupérer le résultat dans une variable.

10. Voici une autre façon de faire, avec quelques raccourcis de programmation :

```
def addition(a, b):
    return a + b

print(addition(6, 6)) # affiche 12
```

Exercice 4

100

Exercice 5

```
def aire_trapeze(a, b, h):
    return (a+b)*h/2
```

Exercice 6

```
from math import sqrt

def aire_triangle(a, b, c):
    p = (a+b+c)/2
    return sqrt(p*(p-a)*(p-b)*(p-c))
```

Exercice 7

```python
def distance_au_carre(x1, y1, x2, y2):
    return (x2-x1)**2+(y2-y1)**2

def est_rectangle(xa, ya, xb, yb, xc, yc):
    ab2 = distance_au_carre(xa, ya, xb, yb)
    ac2 = distance_au_carre(xa, ya, xc, yc)
    bc2 = distance_au_carre(xb, yb, xc, yc)
    if ab2 == ac2+bc2 or ac2 == ab2+bc2 or bc2 == ab2+ac2 :
        return True
    return False

# on vérifie que l'appel suivant vaut True en affichant ce que renvoie la fonction
print(est_rectangle(5, 2, 2, 2, 2, 6))
```

Variante avec le produit scalaire :

```python
def produit_scalaire(ux, uy, vx, vy):
    return ux*vx + uy*vy

def est_angle_droit(xa, ya, xb, yb, xc, yc):
    """ renvoie True ssi l'angle en A est droit"""
    ux = xb-xa
    uy = yb-ya
    vx = xc-xa
    vy = yc-ya
    return produit_scalaire(ux, uy, vx, vy)==0

def est_rectangle(xa, ya, xb, yb, xc, yc):
    return (    est_angle_droit(xa, ya, xb, yb, xc, yc)
             or  est_angle_droit(xc, yc, xa, ya, xb, yb)
             or  est_angle_droit(xb, yb, xc, yc, xa, ya,)
            )


print(est_rectangle(5, 2, 2, 2, 2, 6))
```