Correction du TP 1

Exercice 1 — Moyenne de deux nombres

1. La fonction suivante prend en arguments deux nombres et renvoie leur somme.

```
def somme(a, b):
    """
    Entrées : deux nombres a et b (entiers ou flottants).
    Sortie : la somme de a et b.
    """
    return a + b
```

2. La fonction suivante prend en arguments deux nombres et renvoie leur moyenne.

```
def moyenne(a, b):
    """
    Entrées : deux nombres a et b (entiers ou flottants).
    Sortie : la moyenne (arithmétique) de a et b.
    """
    s = somme(a, b) # Le résultat de la fonction précédente
                    # appliquée à a et b.
    return s / 2
```

Exercice 2 — Échanges

- Le code

```
x = y

y = x
```

proposé n'échange pas les valeurs des variables x et y.

En effet, après la première affectation, la variable x prend la valeur de y (ici, 7) et son ancienne valeur (ici, 2) est écrasée. Au moment de la seconde affectation, la variable x a donc déjà été modifiée, donc la variable y garde sa valeur initiale.

- Pour échanger les valeurs des variables x et y, on peut procéder de deux manières différentes.

  ⋆ **Première méthode : affectation simultanée**

  Une première méthode consiste à affecter au couple (x, y) la valeur du couple (y, x). Les deux variables x et y sont ainsi modifiées simultanément.

```
(x, y) = (y, x)
```

  ⋆ **Seconde méthode : variable auxiliaire**

  Pour échanger les valeurs des deux variables sans utiliser l'affectation simultanée, il faut prendre garde à ne pas perdre la valeur de la première variable que l'on va modifier (pour pouvoir l'affecter ensuite à la seconde variable). On stocke donc la valeur de la première variable dans une variable auxiliaire le temps de l'échange.

```
aux = x # On stocke la valeur de x.
x = y   # La variable x prend la valeur de y.
y = aux # On affecte à y la valeur que l'on a stockée
        # (c'est-à-dire la valeur initiale de x).
```

Exercice 3 — Une bien belle fonction
Écrivons en Python la fonction suivante :

$$x \longmapsto \frac{2x^3 + \ln(x)}{\sqrt{\pi x}}.$$

```
import numpy as np # Pour les fonctions ln et racine carrée,
                   # ainsi que pour le nombre pi.

def bien_belle_fonction(x):
    numerateur = 2 * x ** 3 + np.log(x)
    denominateur = np.sqrt(np.pi * x)
    return numerateur / denominateur
```

Exercice 4 — Évaluation d'une fonction en un point
La fonction suivante prend en arguments une fonction f et un réel x, et évalue la fonction f au point x.

```
def evaluation(f, x):
    """
    Entrées : une fonction f et un flottant x.
    Sortie : la valeur de f(x).
    """
    return f(x)
```

Voici comment l'on procède pour tester la fonction `evaluation` sur nos fonctions préférées.

- Sur la fonction sinus :

```
import numpy as np

evaluation(np.sin, np.pi / 6)
```

- Sur la bien belle fonction de l'exercice précédent :

```
evaluation(bien_belle_fonction, 1.0)
```

- Sur la fonction carré :

```
def carre(x):
    """
    Entrée : un nombre x, entier ou flottant.
    Sortie : le carré de x.
    """
    return x ** 2

evaluation(carre, -1.5)
```

Exercice 5 — Distance entre deux points
La fonction suivante prend en arguments les coordonnées de deux points A et B du plan et renvoie la distance AB, calculée grâce au théorème de Pythagore !

```
import numpy as np # Pour la fonction racine carrée.

def Distance(xA, yA, xB, yB):
    """
    Entrées : quatre réels xA, yA, xB, yB.
    Sortie : distance entre les points du plan
             de coordonnées (xA, yA) et (xB, yB).
    """
    distance = np.sqrt( (xB - xA) ** 2 + (yB - yA) ** 2 )
    return distance
```

Exercice 6 — Parité
Remarquons qu'un entier est pair si et seulement le reste de sa division euclidienne par 2 est nul, et qu'un entier est impair si et seulement si le reste de sa division euclidienne par 2 vaut 1.

Ainsi, étant donné un entier n passé en argument, renvoyer 0 si n est pair et 1 sinon revient à renvoyer le reste de la division euclidienne de n par 2.

```
def parite(n):
    """
    Entrée : un entier n.
    Sortie : 0 si n est pair, 1 sinon.
    """
    return n % 2
```