Correction du TP 2

Exercice 1 — Une fonction définie par morceaux
Écrivons en Python la fonction suivante :

$$f : \mathbb{R} \longrightarrow \mathbb{R}$$
$$x \longmapsto \begin{cases} x^2 & \text{si } x \in]-\pi; 0[\cup]0; \pi] \\ 7 & \text{si } x = 0 \\ 2x+3 & \text{si } x \notin]-\pi; \pi]. \end{cases}$$

```python
import numpy as np # Pour le nombre pi.

def fonction_f(x):
    """
    Entrée : un flottant x.
    Sortie : son image f(x) par la fonction f.
    """
    if (-np.pi < x < 0) or (0 < x <= np.pi):
        y = x ** 2
    elif x == 0:
        y = 7
    else:
        y = 2 * x + 3
    return y
```

Exercice 2 — Distraction
La fonction `mystere` suivante

```python
def mystere(x):
    if x <= 0:
        resultat = -x
    resultat = x
    return resultat
```

prend en argument un nombre (entier ou flottant) x.

Dans le cas où x est négatif, la fonction affecte la valeur $-x$ à une variable locale `resultat`.

Ensuite, quel que soit le signe de x, la fonction affecte la valeur x à la variable `resultat`.
Le test précédent n'a donc en fait aucun intérêt, sinon de nous distraire !

La fonction renvoie ensuite la valeur de `resultat`, qui est donc x.

Ainsi, la fonction `mystere` est la fonction identité $x \longmapsto x$!

Exercice 3 — Années bissextiles
Les fonctions suivantes déterminent si l'année qu'elles prennent en argument est bissextile.

- **Première méthode : tests imbriqués**

```
def bissextile(an):
    """
    Entrée : un entier an.
    Sortie : True si l'année an est bissextile,
             False sinon.
    """
    if an % 400 == 0:
    # Si l'année an est divisible par 400, elle est bissextile.
        return True
    else:
        if an % 4 == 0:
            if an % 100 == 0:
            # Si l'année an n'est pas divisible par 400,
            # mais est divisible par 4 et par 100,
            # alors elle n'est pas bissextile.
                return False
            else:
            # Si l'année an est divisible par 4 mais pas par 100,
            # alors elle est bissextile.
                return True
        else:
        # Si l'année an n'est pas divisible par 4,
        # alors elle n'est pas bissextile.
            return False
```

- **Seconde méthode, plus élégante : renvoi d'un booléen**

```
def bissextile_bis(an):
    """
    Entrée : un entier an.
    Sortie : True si l'année an est bissextile,
             False sinon.
    """
    return ((an % 4 == 0) and (an % 100 != 0)) or (an % 400 == 0)
```

Exercice 4 — À peu presque
La fonction `proche` suivante

```
def proche(a, b, eps):
    diff = np.abs(a - b)
    return diff < eps
```

prend en entrées trois arguments flottants a, b et `eps` (*comme* ε).

La fonction affecte à la variable locale `diff` la valeur absolue de la différence entre a et b :

$$\texttt{diff} = |a - b|,$$

c'est-à-dire la distance entre les réels a et b.

La fonction renvoie alors la valeur du booléen `diff < eps` :

> la fonction `proche` renvoie `True` si la distance entre a et b est strictement plus petite que `eps`, et `False` sinon.

Exercice 5 — Manipulation de booléens
Pour chacun des énoncés, écrivons une fonction qui prend en entrée trois réels a, b et c, et qui renvoie `True` si l'énoncé est vrai et `False` sinon.

1. Les trois réels a, b et c sont nuls.

```
def tous_nuls(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si a, b et c sont tous nuls, False sinon.
    """
    resultat = (a == 0) and (b == 0) and (c == 0)
    return resultat
```

   Il est également possible en Python d'enchainer les tests d'égalités (comme en mathématiques), de la manière suivante.

```
def tous_nuls_bis(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si a, b et c sont tous nuls, False sinon.
    """
    resultat = (a == b == c == 0)
    return resultat
```

2. Au moins l'un des trois nombres a, b ou c est nul.

```
def un_nul(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si l'un des trois réels a, b et c est nul,
             False sinon.
    """
    return (a == 0) or (b == 0) or (c == 0)
```

3. Les trois réels a, b et c sont non nuls.

```
def tous_non_nuls(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si a, b et c sont tous non nuls,
             False sinon.
    """
    resultat = (a != 0) and (b != 0) and (c != 0)
    return resultat
```

4. Les réels a, b et c ne sont pas tous nuls.

```
def non_tous_nuls(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si a, b et c sont non tous nuls,
             False sinon.
    """
    return not(tous_nuls(a, b, c))
```

5. Les trois réels a, b et c ne sont pas tous égaux.

```
def non_tous_egaux(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si a, b et c ne sont pas tous égaux,
             False sinon.
    """
    resultat = not((a == b) and (b == c))
    return resultat
```

En enchainant les tests d'égalité, la fonction s'écrit aussi de la manière suivante.

```
def non_tous_egaux_bis(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si a, b et c ne sont pas tous égaux,
             False sinon.
    """
    resultat = not(a == b == c)
    return resultat
```

6. Un seul des trois réels a, b et c est non nul.

```
def un_seul_non_nul(a, b, c):
    """
    Entrées : trois réels a, b et c.
    Sortie : True si un seul des trois réels a, b et c
             est non nul, False sinon.
    """
    resultat = ((a != 0) and (b == c == 0))
            or ((b != 0) and (a == c == 0))
            or ((c != 0) and (a == b == 0)) # Sur une seule ligne.
    return resultat
```

Exercice 6 — Le plus grand impair

Commençons par écrire une fonction qui prend en arguments deux nombres et qui renvoie le plus petit des deux.

```
def minimum(a, b):
    """
    Entrées : deux nombres a et b (entiers ou flottants).
    Sortie : le minimum de a et b.
    """
    if a < b:
        return a
    else:
        return b
```

Écrivons une fonction qui prend en arguments trois nombres entiers x, y, z et qui renvoie 0 si aucun des trois nombres n'est impair, et le plus grand nombre impair parmi x, y, z sinon.

```
def plus_grand_impair(x, y, z):
    """
    Entrées : trois entiers x, y et z.
    Sortie : 0 si aucun des trois nombres x, y, z n'est impair,
             le plus grand nombre impair parmi x, y, z sinon.
    """
    valeur_petite = minimum(x, minimum(y, z)) - 1
    # Le nombre valeur_petite vaut un de moins
    # que le minimum des trois entiers x, y, z,
    # donc est strictement plus petit
    # que tous les entiers entrés.
    max_impair = valeur_petite
    # Pour l'instant, on fixe le maximum à cette petite valeur.
    # Dès qu'on rencontre un entier impair plus grand,
    # il se substitue au maximum courant.
    if (x % 2 == 1):
        max_impair = x
    if (y % 2 == 1) and (y > max_impair):
        max_impair = y
    if (z % 2 == 1) and (z > max_impair):
        max_impair = z
    # Si le maximum est resté à la petite valeur,
    # c'est qu'aucun des entiers n'était impair.
    if max_impair == valeur_petite:
        return 0
    else:
        return max_impair
```