PRG. MAP. Exercices. Corrigé

***Exercice 1** : typage de paramètres et valeur de retour dans une fonction*

```
def f1(t : list) -> int :
    return t[0] + 1

def f2(x : float) -> str :
    return str(4.5*x)

def f3(p : tuple) -> float :
    x, y = p
    return 2*x + y
```

```
def f4(s : list) -> NoneType:
    s.append(40)

def f5(d : dict,s : str) -> int:
    if s != "yop" :
        d[s] += 1
    return d[s]
```

***Exercice 2** : test d'une fonction*

```
def test(str1,str2) :
    n = len(str1)

    # Vérification de la longueur
    # de la chaîne `str2`
    assert len(str2) == n,"Chaînes de longueurs différentes"
    # Vérification caractère par caractère
    for i in range(n) :
        assert str1[i] == str2[n - 1 - i],"Inversion non valide"

str1 = "bonjour" # Chaîne de caractères de base
str2 = "ruojnob" # Application de la fonction "miroir"

test(str1,str2)
```

(Tout se passe bien ici, str2 est bien la chaîne inverse de str1)

```
11 str1 = "bonjour" # Chaîne de caractères de base
12 str2 = "ruojnoB" # Application de la fonction "miroir"
13
14 test(str1,str2)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-4-1732d4f19cca> in <module>
     12 str2 = "ruojnoB" # Application de la fonction "miroir"
     13
---> 14 test(str1,str2)

<ipython-input-4-1732d4f19cca> in test(str1, str2)
      7     # Vérification caractère par caractère
      8     for i in range(n) :
----> 9         assert str1[i] == str2[n - 1 - i],"Inversion non valide"
     10
     11 str1 = "bonjour" # Chaîne de caractères de base

AssertionError: Inversion non valide
```

(Message d'erreur d'assertion ici, le caractère « B » ne correspond pas au caractère « b » !)

Remarque : vérifier que les deux chaînes (str1 et str2 ici) ont bien la même longueur permet d'éviter un éventuel plantage lors de la boucle `for`.

***Exercice 3** : ajout de secondes à un temps*

```python
def secondes(heure, minute, seconde, sec) :
    # Contrôle des plages de valeurs
    assert heure <= 23 and heure >= 0 and minute < 60 \
    and minute >= 0 and seconde < 60 and seconde >= 0

    n_secondes = seconde + sec
    # 60 secondes = 1 minute
    n_minutes = minute + sec // 60
    # 60 secondes au maximum
    n_secondes = n_secondes % 60
    # 60 minutes = 1 heure
    n_heures = heure + n_minutes // 60
    # 60 minutes au maximum
    n_minutes = n_minutes % 60
    # 24 heures au maximum
    n_heures = n_heures % 24


    return n_heures, n_minutes, n_secondes
```

Jeu de tests

```python
print(secondes(15,47,10,3603)) # Attendu : (16,47,13)
print(secondes(15,47,10,-4700)) # Attendu : (14,28,50)
print(secondes(23,17,30,3672)) # Attendu : (00,18,42)
```

```
(16, 47, 13)
(14, 28, 50)
(0, 18, 42)
```

***A noter** : l'antislash « \ » permet de revenir à la ligne en langage Python sans interrompre une instruction.*