Tester automatiquement ses fonctions avec `doctest`

Chap. 5

Table des matières

1 Rappel 1

2 Première utilisation du module `doctest` 2

2.1 À quoi sert le module `doctest` ? 2
2.2 Comment rendre l'analyse de la valeur retournée par une fonction automatique ? 3
2.3 Que faire lorsqu'on souhaite prendre en compte la mauvaise utilisation d'une fonction ? 4

3 À retenir 5

4 Exercices 5

---

1 Rappel

Au cours du chapitre 2, nous avons expliqué pourquoi *la définition d'une fonction doit toujours être accompagnée de l'écriture de sa spécification*. Cette dernière permet :

→ à tout utilisateur de la fonction de déterminer facilement

- quel traitement celle-ci réalise ;
- comment il faut l'utiliser (nombre d'arguments par exemple) ;
- quelles conditions il est nécessaire de respecter pour l'utiliser.

→ au programmeur de cette fonction de préciser

- le nombre et le type de ses paramètres ;
- le type de la valeur retournée ;
- le traitement exécuté, accompagné de quelques exemples.

Spécification = Signature + Documentation

Exemple.

```
def kelvin_vers_celsius(t: float) -> float:
```

```
    """ Convertit la température donnée en kelvin en une température θ
    en degrés celsius. Relation : θ = t - 273.15.

    >>> kelvin_vers_celsius(0)
    -273.15
    >>> kelvin_vers_celsius(273.15)
    0.0
    """
    offset = -273.15
    return t + offset
```

La spécification d'une fonction peut être obtenue à l'aide de la fonction `help` :

```
>>> help(kelvin_vers_celsius)
  Help on function kelvin_vers_celsius in module __main__:

  kelvin_vers_celsius(t: float) -> float
     Convertit la température donnée en kelvin en une température θ
     en degrés celsius. Relation : θ = t - 273.15.

     >>> kelvin_vers_celsius(0)
     -273.15
     >>> kelvin_vers_celsius(273.15)
     0.0
```

L'objectif de ce chapitre est de démontrer que l'écriture d'une spécification présente encore bien plus d'avantages !

2 Première utilisation du module `doctest`

2.1 À quoi sert le module `doctest` ?

À faire.

1. Taper le code *complet* de la définition de la fonction `kelvin_vers_celsius` dans la zone de définition des fonctions du logiciel `Thonny`.

2. Déplacer le début du code précédent au niveau de la ligne 3 et écrire, à la ligne 1, l'instruction :

   ```
   import doctest
   ```

   Enregistrer le fichier contenant le code précédent.

3. Charger le fichier dans l'interpréteur (touche `F5` du clavier). La fonction peut être appelée, rien de particulier n'apparaît pour l'instant.

   ```
   >>> kelvin_vers_celsius(373.15)
   100.0
   ```

4. Entrer et exécuter l'instruction suivante :

```
>>> doctest.testmod()
TestResults(failed=0, attempted=2)
```

La fonction `testmod` du module `doctest` semble avoir effectué deux tests (`attempted=2`) sans avoir rencontré le moindre problème (`failed=0`).

Mais quels tests a-t-elle effectué ?

5. Entrer et exécuter l'instruction suivante :

```
>>> doctest.testmod(verbose = True)
Trying:
  kelvin_vers_celsius(0)
Expecting:
   -273.15
ok
Trying:
   kelvin_vers_celsius(273.15)
Expecting:
   0.0
ok
...
```

La fonction `testmod` du module `doctest` a

- lu la documentation de la fonction `kelvin_vers_celsius` à la recherche de lignes débutant par `>>>` ;
- exécuté les instructions de ces lignes ;
- a vérifié que le résultat retourné correspond bien à ce qui est écrit dans cette documentation.

6. Modifier maintenant, dans la définition de la fonction `kelvin_vers_celsius`, la valeur `0.0` par `1.0`. Charger à nouveau le fichier dans l'interpréteur (touche `F5` du clavier) et relancer la commande du point 5 ci-dessus.

   Que retourne la fonction `testmod` maintenant ? Ce message décrit-il précisément le problème ?

La fonction `testmod` *du module* `doctest` *analyse la valeur retournée par une fonction à partir d'exemples données dans sa documentation.*

2.2 Comment rendre l'analyse de la valeur retournée par une fonction automatique ?

À faire.

1. Effacer l'instruction « `import doctest` » de la ligne 1 de la zone de définition des fonctions du logiciel `Thonny`.

2. Inclure, à la fin de cette zone (**après la définition de toutes les fonctions**), le bloc d'instructions suivant :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

3. Charger à nouveau le fichier dans l'interpréteur (touche `F5` du clavier).

   La fonction `testmod` est lancée automatiquement et retourne la même erreur qu'au point 6 de la section 2.1 ci-dessus.

4. Modifier maintenant, dans la documentation de la fonction `kelvin_vers_celsius`, la valeur `1.0` par `0.0`. Charger à nouveau le fichier dans l'interpréteur (touche `F5` du clavier).

   A-t-on la moindre indication que la fonction `testmod` a été lancée ?

2.3 Que faire lorsqu'on souhaite prendre en compte la mauvaise utilisation d'une fonction ?

À faire.

1. Remplacer la définition de la fonction `kelvin_vers_celsius` par celle-ci :

```
def kelvin_vers_celsius(t: float) -> float:
    """ Convertit la température donnée en kelvin en une température θ
    en degrés celsius. Relation : θ = t - 273.15. ERREUR si t est négative.

    >>> kelvin_vers_celsius(0)
    -273.15
    >>> kelvin_vers_celsius(273.15)
    0.0
    """
    if t < 0:
        raise ValueError("Température en kelvin doit être positive !")

    offset = -273.15
    return t + offset
```

   Quelle est la différence dans le comportement de la fonction ce code introduit-il ?

2. Pour l'instant la documentation de la fonction ne comporte aucun exemple illustrant ce à quoi il faut s'attendre si on appelle la fonction avec un nombre négatif.

   a. Charger le fichier dans l'interpréteur (touche `F5` du clavier).

   b. Écrire et exécuter l'instruction

```
>>> kelvin_vers_celsius(-5)
```

   On obtient

```
Traceback (most recent call last):
...
ValueError: Température en kelvin doit être positive !
```

c. Ce sont ces lignes qu'il faut ajouter à la documentation de la fonction

```
>>> kelvin_vers_celsius(-5)
Traceback (most recent call last):
...
ValueError: Température en kelvin doit être positive !
```

Compléter la documentation de la fonction et la tester.

3 À retenir

Pour chacune des fonctions, il faut

1. incorporer à sa documentation des exemples pertinents d'utilisation de ces fonctions ;
2. ajouter le bloc d'instruction suivant à la fin du fichier

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

4 Exercices

1. Définir le prédicat `est_egal` dont une partie de la spécification est :

```
def est_egal(a: float, b: float) -> bool:
    """
    Teste l'égalité de 2 floats a et b : abs(a - b) <= epsilon avec
    epsilon = 1e-7

    >>> est_egal(0, 0.0)
    xxx
    >>> est_egal(1/49*49, 1)
    xxx
    """
```

Remplacer les `xxx` de façon à ce que la fonction `testmod` accepte votre définition.

2. Définir la fonction `inverse` dont une partie de la spécification est :

```
def inverse(x: float) -> float:
""" Retourne l'inverse du nombre x.
ERREUR si x est nul.

>>> inverse(0)
Traceback (most recent call last):
...
ZeroDivisionError: Division par 0 !!!
>>> inverse(5)
xxx
"""
```

Remplacer les `xxx` de façon à ce que la fonction `testmod` accepte votre définition.