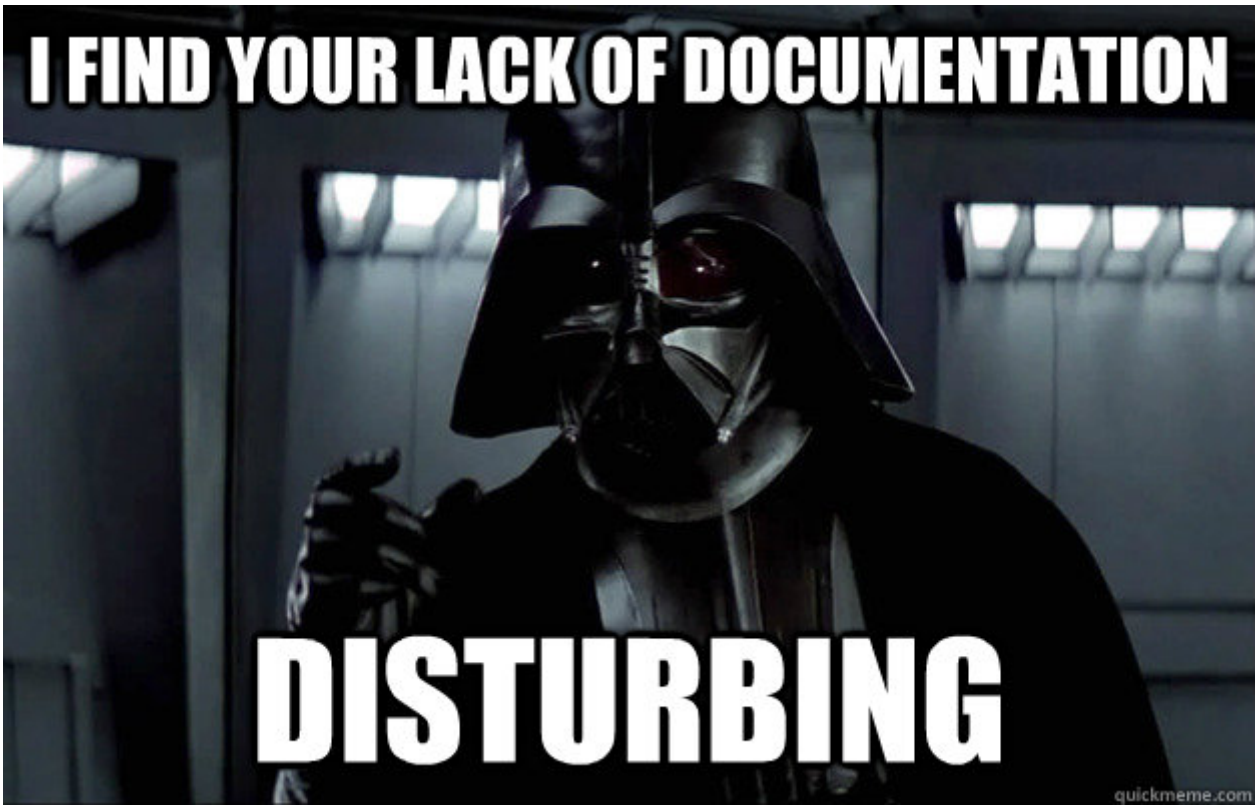
T6.4 Spécification

Commençons par une devinette: que fait la fonction suivante:

Script Python

```python
def doublon(l, c):
    n = 0
    for e in c:
        if e == l:
            n += 1
    return n > 1
```



Mieux?

Script Python

```
def verifie_doublon(l: str, c: str) -> bool:
    '''
    Vérifie si la chaîne c contient au moins deux occurences
    de la chaîne l
    '''
    nb_occurences = 0
    for element in c:
        if element == l:
            nb_occurences += 1
    return nb_occurences > 1
```

1. 6.4.1 Spécifier et documenter

Spécification

De manière générale la **spécification** est un ensemble de d'exigences à satisfaire par un produit ou un service.

En programmation, **spécifier** un programme/une fonction revient à décrire explicitement ce qu'il/elle doit faire et dans quelles conditions.

Pour cela:

- on décrit ce que fait la fonction dans une documentation appelée **docstring**, placée au début du corps de la fonction, écrite sur plusieurs lignes entre triple quotes;
- on utilise des noms de variables explicites (voir T6.1.1);
- on précise le type de(s) paramètre(s), et éventuellement les conditions d'utilisations d ela fonction: on parle de **préconditions**;
- on précise le type de la valeur renvoyée par la fonction: on parle de **postconditions**;
- ces types apparaissent dans la docstring ou en *annotations de types* (depuis Python 3.5) comme dans l'exemple ci-dessus;
- on intègre éventuellement des exemples d'utilisation qui peuvent servir de tests.

La docstring est alors disponible en utilisant la fonction `help`:

Script Python

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

```

```
verifie_doublon(l: str, c: str) -> bool
    Vérifie si la chaîne c contient au moins deux occurences
    de la chaîne l

>>>
```

Intérêt

- programmer : documenter **avant** d'écrire le code donne un objectif clair;
- relire : les programmes complexes sont difficiles à comprendre. La documentation simplifie cette étape;
- collaborer : travailler à plusieurs demande de l'organisation et une documentation claire est indispensable.

PEP8 & Zen

Pour une meilleure lecture du code et une meilleure communication entre les différents collaborateurs d'un projet, il est donc nécessaire de prendre de bonnes habitudes de programmation et de respecter des conventions d'écriture:

- La PEP8 (Python Enhancement Proposal 8) regroupe les bons usages
- Le Zen Python :

Script Python

```
>>> import this
```

2. 6.4.2 Exemples

Exemple sans tests

La fonction `maximum` vue aux chapitres précédentes:

Script Python

```python
def maximum(n1, n2):
    if n1 > n2:
        return n1
    else:
        return n2
```

devient:

Script Python

```python
def maximum(n1: int, n2: int) -> int:
    '''
    Renvoie l'entier maximum entre n1 et n2
    n1 et n2 : deux entiers
    '''
    if n1 > n2:
        return n1
    else:
        return n2
```

Exemple avec tests

Voici la fonction `bissextile` spécifiée et documentée. On peut y ajouter des exemples/tests.

Script Python

```
def is_leap(year: int) -> bool:
    '''
    Renvoie True si l'entier year correspond à une année bissextile
    et False sinon.
    >>> is_leap(2021)
    False
    >>> is_leap(2020)
    True
    >>> is_leap(2000)
    True
    >>> is_leap(1900)
    False
    '''
    if year%4 == 0 and year%100 != 0:
        return True
    elif year%400 == 0:
        return True
    else:
        return False
```

On peut ensuite lancer les tests à l'aide du module `doctest` :

Script Python

```
>>> import doctest
>>> doctest.testmod()
TestResults(failed=0, attempted=4)
>>>
```

Testez cette fonction comme ci-dessus, puis en passant en argument `verbose=True` à la fonction `testmod`.

3. 6.4.3 Exercices

Exercice 1

Énoncé

Documenter les fonctions `compte_voyelles` et `fizzbuzz` des chapitres précédents.

Correction