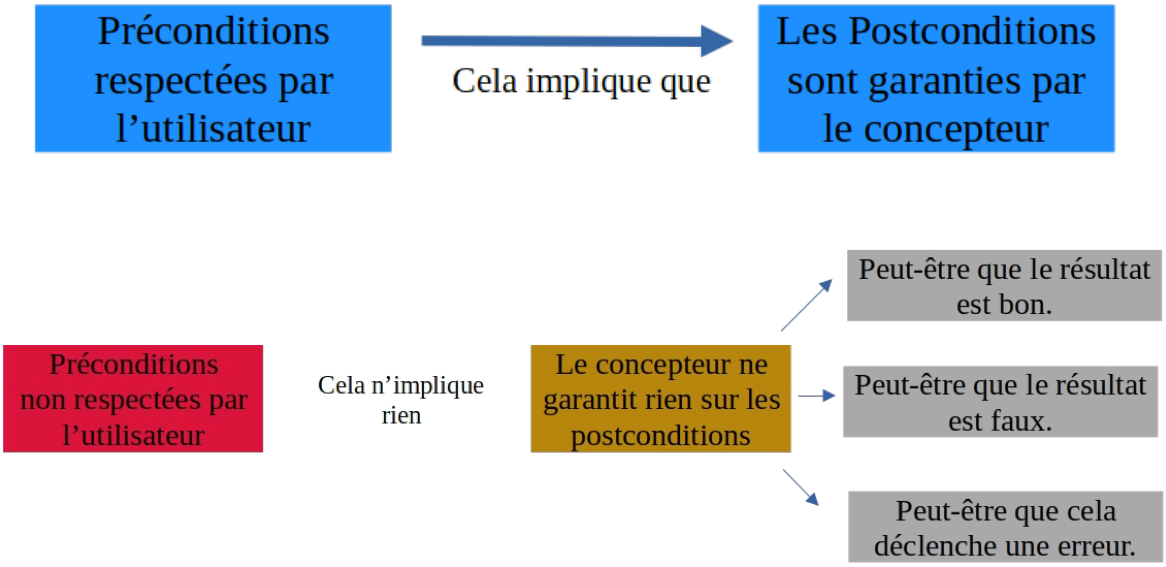
Python 14 – Documenter et Tester

Introduction – Contrat de confiance

Programmer consiste à passer des accords de confiance entre l'utilisateur d'une fonction et le concepteur de cette même fonction :



PRECONDITIONS : conditions supplémentaires EN PLUS du simple type sur les arguments que l'utilisateur a le droit d'envoyer à la fonction.

POSTCONDITIONS : conditions que le concepteur garantit sur la sortie EN PLUS du simple type de la réponse.

On retrouve donc la même notion qu'avec l'équivalence logique qui permet d'éviter de synthétiser les relations sans avoir à noter trop de choses :

$$A \rightarrow B$$

veut dire

A implique B

qui veut dire

Si la propriété A est vraie alors la propriété B est vraie
Si A est fausse alors on ne peut rien dire sur la propriété B(elle peut être vraie ou fausse)

I – Rappel : documenter

Pour permettre au chef d'équipe, au développeur et à l'utilisateur de la fonction de communiquer correctement, il est important de rajouter une documentation sous forme d'un string : un **DOCSTRING** commençant par """ et finissant par """ qui contient toutes les informations nécessaires à la bonne compréhension de l'utilisation permise et du but de la fonction.

En Python, on peut fournir
→ soit une documentation longue (voir annexe)
→ soit une documentation plus courte indiquant les types directement sur le prototype.

Cette documentation permet également de fournir au créateur de la fonction, toutes les informations nécessaires pour qu'il réalise correctement sa tâche. Elle contient donc normalement également des exemples d'utilisation, comme si on tapait le tout dans la console.

```
1       def les_mots(s:'str CONTENANT AU MOINS UN MOT') -> 'list[str] NON VIDE':
2           """Renvoie un tableau contenant les MOTS NON VIDES contenus dans s
3
4           ..exemples..
5           >>> t = les_mots("Bonjour, à tous !")
6           >>> t
7           ['Bonjour', 'à', 'tous']
8
9           """
10
11          t = s.split(" ")            # tableau obtenu en utilisant ESPACE comme séparateur
12          for i in range(len(t)):           # Pour chaque indice i (on va modifier donc i)
13              for ponctuation in "?!.,;:":              # Pour chaque ponctuation
14                  t[i] = t[i].replace(ponctuation, "")   # Supprime cette ponctuation
15
16          # Crée un tableau contenant chaque mot de t si ce mot n'est pas un string vide
17          rep = [mot for mot in t if mot != ""]
18          return rep
```

Sur cet exemple,
PRECONDITION : le concepteur demande d'envoyer un string contenant au moins un mot
POSTCONDITION : le concepteur garantit quon obtiendra un tableau NON VIDE de strings.

II – Tester avec une fonction de test et des assert

Documenter est important.
Tester aussi, mais c'est long manuellement.
On réalise des **tests automatisés** en utilisant le mot-clé **`assert`** qui vérifie la propriété fournie.
Si elle est vrai : Python continue.
Si elle est fausse : **`AssertionError`**

On peut donc placer cela dans le programme de façon à lancer automatiquement les tests dès que le programme est lancé : on détectera tout de suite n'importe quelle erreur due à une modification sur un programme qui fonctionnait auparavant.

```
1       # les lignes de la partie précédentes
..
19
20      def tester():
21          # Test 1
22          reponse = les_mots("Bonjour, à tous !")
23          assert reponse == ['Bonjour', 'à', 'tous']
24          # Test 2
25          reponse = les_mots("Bonjour ...")
26          assert reponse == ['Bonjour']
27
28      tester()
```

Or, la réalisation des tests ralentissent le programme. Ils sont présents par exemple dans le module **`math`** ou le module **`turtle`**, mais vous n'avez pas envie de lancer les tests lorsque vous utilisez **`math.py`** comme un module ? Dans ce cas, on peut demander poliment à Python de ne lancer les tests qui se on lance notre programme en tant que programme principal et pas en tant que code importé.

```
1       # les lignes de la partie précédentes
..
28      if __name__ == "__main__":   # Si ce programme est bien le prog. Principal
29          tester()
```

III – Tester avec le module doctest

Reste qu'on doit donc :
→ fournir des exemples d'utilisation dans le DOCSTRING
→ fournir des tests dans les fonctions de test.

On en vient donc à taper deux fois les mêmes choses ou presque.
Python simplifie le travail de développement en fournissant un module nommé **`doctest`** qui permet d'utiliser les exemples fournis dans la documentation pour réaliser des tests : doc → test.

Voici un exemple d'utilisation automatisée. Si les tests donnent satisfaction, Python n'indiquera rien. Si certains tests ne fonctionnent pas, on aura un message d'alerte.

```
 1        def les_mots(s:'str CONTENANT AU MOINS UN MOT') -> 'list[str] NON VIDE':
 2            """Renvoie un tableau contenant les MOTS NON VIDES contenus dans s
 3
 4            ..exemples..
 5            >>> t = les_mots("Bonjour, à tous !")
 6            >>> t
 7            ['Bonjour', 'à', 'tous']
 8
 9            """
10
11            t = s.split(" ")            # tableau obtenu en utilisant ESPACE comme séparateur
12            for i in range(len(t)):           # Pour chaque indice i (on va modifier donc i)
13                for ponctuation in "?!.,;:":              # Pour chaque ponctuation
14                    t[i] = t[i].replace(ponctuation, "")   # Supprime cette ponctuation
15
16            # Crée un tableau contenant chaque mot de t si ce mot n'est pas un string vide
17            rep = [mot for mot in t if mot != ""]
18            return rep
19
20
21        if __name__ == "__main__":   # Si ce programme est bien le prog. Principal
22            import doctest           # on importe le module doctest
23            doctest.testmod()        # on lance la fonction testmod() de ce module
```

Si le programme est **lancé directement**, les **lignes 22 et 23 s'exécutent** et la fonction **`testmod()`** va donc lire la documentation et les utiliser pour réaliser des tests, comme si vous avez tapé les assertions précédentes.

Si le programme est en réalité **importé** par un autre programme, son nom interne n'est pas **"__main__",** la condition derrière le if est **`False` et on ne lance donc pas les lignes 22 et 23.**