Spécification et tests

Fiche de synthèse

1 Prototype

Définition 1 **(Prototype)** Le **prototype** d'une fonction est la première ligne de sa définition : le nom de la fonction, ses paramètres avec leurs types, et le type de la valeur renvoyée.

```
def aire_rectangle(longueur: float,
                   largeur: float) -> float:
```

Cette fonction prend deux paramètres de type `float` et renvoie un `float`.

2 Docstring et doctests

Définition 2 **(Docstring)** La **docstring** est la chaîne de documentation placée juste après le prototype. Elle donne une description brève, les paramètres et leurs types, la valeur renvoyée, des exemples et, si besoin, les préconditions.

```
def aire_rectangle(longueur: float,
                   largeur: float) -> float:
    """Calcule l'aire d'un rectangle.

    Paramètres :
        longueur (float) : la longueur.
        largeur (float) : la largeur.
    Renvoie :
        float : l'aire du rectangle.
    Exemples :
        >>> aire_rectangle(10.0, 5.0)
        50.0
        >>> aire_rectangle(0.0, 10.0)
        0.0
    """
    assert longueur >= 0 and largeur >= 0
    return longueur * largeur
```

Un **doctest** est un exemple écrit dans la docstring : une ligne `>>>` suivie de l'appel, puis le résultat attendu. On les exécute en ajoutant à la fin du fichier :

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

Avec `verbose=True`, chaque test est affiché, réussi ou non. On choisit des exemples variés : des cas courants et des cas limites, comme une longueur nulle.

3 Préconditions

Définition 3 **(Précondition)** Une **précondition** est une condition que les paramètres doivent vérifier pour que la fonction donne un résultat correct.

```
def racine_carree(x: float) -> float:
    """Renvoie la racine carrée de x."""
    assert x >= 0, "x doit être positif ou nul"
    return x ** 0.5
```

`assert condition, "message"` ne fait rien si la condition est vraie ; sinon, le programme s'arrête sur l'erreur `AssertionError`, suivie du message. `racine_carree(-4)` s'arrête ainsi. Pour une division, la précondition est un diviseur différent de 0.

4 Postconditions

Définition 4 **(Postcondition)** Une **postcondition** est une condition que vérifie le résultat quand les préconditions sont respectées : ce que la fonction garantit. Elle figure dans la section `Renvoie :` de la docstring.

```
def note_sur_20(points, total):
    assert total > 0 and 0 <= points <= total
    note = points / total * 20
    assert 0 <= note <= 20
    return note
```

La postcondition se vérifie avec `assert`, juste avant le `return` : une formule fausse, comme `total / points * 20`, arrête le programme dans la fonction elle-même.

5 Jeux de tests

Un **jeu de tests** est un ensemble d'appels dont on connaît le résultat attendu : des doctests, ou une instruction `assert` par appel.

```
assert aire_rectangle(7.0, 3.0) == 21.0
assert aire_rectangle(0.0, 10.0) == 0.0
```

La qualité des tests compte autant que leur nombre : cas courants et cas limites. Le succès d'un jeu de tests ne garantit pas la correction : `longueur * longueur` réussit le test d'une longueur nulle.

6 Bibliothèques

Une **bibliothèque**, ou module, regroupe des fonctions réutilisables, comme `math` ou `random` : inutile de les réécrire.

```
import math
print(math.sqrt(16))      # 4.0

from math import sqrt, pi
print(sqrt(25))           # 5.0

import math as m
print(m.sqrt(36))         # 6.0
```

`import math` importe tout le module, et l'on écrit `math.` devant chaque fonction ; `from math import sqrt` importe une seule fonction, écrite sans préfixe ; `import math as m` donne au module un nom plus court.

À retenir.

- Une fonction bien écrite a un prototype annoté et une docstring complète.
- Les doctests vérifient les exemples de la docstring ; on y met des cas courants et des cas limites.
- `assert` vérifie les préconditions au début de la fonction, et la postcondition avant le `return`.
- Un jeu de tests réussi ne prouve pas que la fonction est correcte.