

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

# Tests unitaires

Christophe Viroulaud

Terminale - NSI

**Algo 16**

Une tâche importante d'un programmeur est de garantir le bon comportement de son code, dans tous les cas de figures.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Comment mettre en place des tests unitaires ?

1. **Nécessité de tester**
2. Repérer les cas limites
3. Garantir le bon comportement

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

# Nécessité de tester

```
1 def images(deb, fin: int) -> list:
2     """
3     calcule les images d'une fonction
4     """
5     tab = []
6     for i in range(deb, fin):
7         tab.append(f(i))
8     return tab
9
10 def f(x: int) -> float:
11     return 1/x
```

```
1 >>> images(1, 5)
2 [1.0, 0.5, 0.3333333333333333, 0.25]
3 >>> images(0, 5)
4 ZeroDivisionError: division by zero
```

## Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

- ▶ Un projet est composé de nombreux modules, eux-mêmes découpés en plusieurs fonctions.
- ▶ Une fonction **f1** peut utiliser le résultat renvoyé par une fonction **f2** qui utilise le résultat de **f3**...

## À retenir

Chaque fonction doit être testée individuellement, pour chaque cas de figure.

1. Nécessité de tester
2. Repérer les cas limites
3. Garantir le bon comportement

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

# Repérer les cas limites

## Observation

La fonction `f` impose des contraintes mathématiques.

```
1 def images(deb, fin: int) -> list:
2     """
3     calcule les images d'une fonction
4     """
5     tab = []
6     for i in range(deb, fin):
7         tab.append(f(i))
8     return tab
9
10 def f(x: int) -> float:
11     return 1/x
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive



Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

```
1 def f(x: int) -> float:  
2     assert x != 0, "pas de division par zéro"  
3     return 1/x
```

Code 1 – Mettre en place des **assertions**

```
1 def f(x: int) -> float:  
2     if x == 0:  
3         raise Exception("pas de division par 0")  
4     return 1/x
```

Code 2 – Lever une **exception** personnalisée

## À retenir

Cette notion est hors-programme en Terminale.

1. Nécessité de tester
2. Repérer les cas limites
3. **Garantir le bon comportement**
  - 3.1 Mise en évidence
  - 3.2 Automatiser les tests
  - 3.3 Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

**Garantir le bon comportement**

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

# Garantir le bon comportement - Mise en évidence

## À retenir

Même sans erreur mathématique, une fonction peut ne pas renvoyer le résultat attendu.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

**Mise en évidence**

Automatiser les tests

Tester de manière exhaustive

**Activité 1** : La fonction `est_complet` doit renvoyer `True` si la matrice d'adjacence représente un graphe complet.

```
1 def est_complet(mat: list) -> bool:
2     """
3     vérifie si chaque sommet est relié à tous
4     les autres (sauf lui même)
5     """
6     for ligne in range(len(mat)):
7         for col in range(len(mat)):
8             if ligne != col and mat[ligne][col] == 0:
9                 return False
10    return True
```

Proposer un cas de figure où la fonction n'a pas le bon comportement.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

# Correction

```
1 mat = [ [1, 1, 1, 1, 1],  
2         [1, 0, 1, 1, 1],  
3         [1, 1, 0, 1, 1],  
4         [1, 1, 1, 0, 1],  
5         [1, 1, 1, 1, 0]]
```

Code 3 – La fonction renvoie **True** avec cette matrice.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

**Mise en évidence**

Automatiser les tests

Tester de manière exhaustive

```
1 def est_complet(mat: list) -> bool:
2     """
3     vérifie si chaque sommet est relié à tous
4     les autres (sauf lui même)
5     """
6     for ligne in range(ordre(mat)):
7         for col in range(ordre(mat)):
8             if (ligne != col and mat[ligne][col] == 0) or \
9                 (ligne == col and mat[ligne][col] == 1):
10                 return False
11     return True
```

Code 4 – La fonction correcte

1. Nécessité de tester
2. Repérer les cas limites
3. Garantir le bon comportement
  - 3.1 Mise en évidence
  - 3.2 **Automatiser les tests**
  - 3.3 Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

**Automatiser les tests**

Tester de manière exhaustive



## À retenir

L'objectif de la mise en place des tests unitaires est de proposer une structure qui garantira le bon comportement de chaque fonction tout au long de l'évolution du projet.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

**Automatiser les tests**

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

**Automatiser les tests**

Tester de manière exhaustive

Les tests doivent être :

- ▶ exhaustifs (possible?),
- ▶ évolutifs en fonction des modifications du projet,
- ▶ exécutable de manière automatisée.

La bibliothèque Python `unittest` permet de réaliser des tests unitaires. La stratégie consiste à créer une classe qui hérite de `unittest.TestCase` puis d'écrire une méthode pour chaque fonction à tester.

```
1 import unittest
2 from mathematiques import f # fonction
   inverse
3
4 class Tests(unittest.TestCase):
5
6     def test_inverse(self):
7         self.assertTrue(f(1.) == 1.)
8         self.assertTrue(f(-1) == -1.)
9
10
11 if __name__=="__main__":
12     unittest.main()
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

```
1 import unittest
2 from mathematiques import f # fonction inverse
3
4 class Tests(unittest.TestCase):
5
6     def test_inverse(self):
7         self.assertTrue(f(1.) == 1.)
8         self.assertTrue(f(-1) == -1.)
9
10 if __name__=="__main__":
11     unittest.main()
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

## Activité 2 :

1. Créer un fichier `mes_tests.py`
2. Écrire le code précédent (adapter le nom du fichier).

### Activité 3 :

```
1 def test_inverse_erreur(self):  
2     # vérifie que la valeur 0 provoque une  
   erreur  
3     with self.assertRaises(ZeroDivisionError):  
4         f(0)
```

Code 5 – Tester une erreur

Ajouter la méthode précédente.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

**Automatiser les tests**

Tester de manière exhaustive

Il existe plusieurs assertions testables :

Méthode	Vérification
<code>assertEqual(a, b)</code>	<code>a == b</code>
<code>assertIs(a, b)</code>	<code>a is b</code>
<code>assertIn(a, b)</code>	<code>a in b</code>

Il est possible de définir un environnement de test pour se rapprocher des conditions d'utilisation.

```
1 class Tests(unittest.TestCase):
2
3     def setUp(self):
4         # on définit des attributs...
5         self.val = 1
6
7     def test_inverse(self):
8         # ...utilisables dans les tests
9         self.assertTrue(f(self.val) == 1.)
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

## Activité 4 : Proposer un jeu de tests pour valider la fonction `est_complet`.

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

nière

```
1 def est_complet(mat: list) -> bool:
2     """
3     vérifie si chaque sommet est relié à tous
4     les autres (sauf lui même)
5     """
6     for ligne in range(len(mat)):
7         for col in range(len(mat)):
8             if (ligne != col and mat[ligne][col] == 0) or \
9                 (ligne == col and mat[ligne][col] == 1):
10                 return False
11     return True
```



# Correction

```
1 class Tests(unittest.TestCase):
2
3     def setUp(self):
4         # solution non exhaustive
5         self.mat_ok = [[0, 1, 1, 1, 1],
6                        [1, 0, 1, 1, 1],
7                        [1, 1, 0, 1, 1],
8                        [1, 1, 1, 0, 1],
9                        [1, 1, 1, 1, 0]]
10        self.mat_no = [[1, 1, 1, 1, 1],
11                       [1, 0, 1, 1, 1],
12                       [1, 1, 0, 1, 1],
13                       [1, 1, 1, 0, 1],
14                       [1, 1, 1, 1, 0]]
15
16     def test_complet(self):
17         self.assertTrue(est_complet(self.mat_ok))
18         self.assertFalse(est_complet(self.mat_no))
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

**Automatiser les tests**

Tester de manière exhaustive

## À retenir

Pour prendre en compte tous les cas de figures, le nombre de tests peut être important.

1. Nécessité de tester
2. Repérer les cas limites
3. Garantir le bon comportement
  - 3.1 Mise en évidence
  - 3.2 Automatiser les tests
  - 3.3 Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

# Tester de manière exhaustive

## À retenir

Pour tester un code il faut prendre en compte tous les cas de figures.

```
1 def ma_fonction(a: int, b: int) -> tuple:
2     if a != 0:
3         b = 5 - a
4     else:
5         b = b - a
6     if b > 3:
7         b = b//a
8     else:
9         b = 0
10    return (a, b)
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

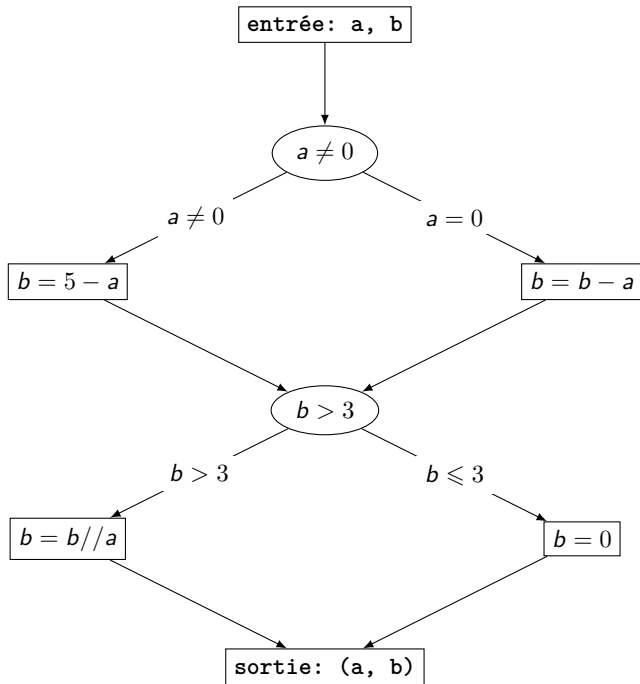
Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive



Nécessité de tester

Repérer les cas limites

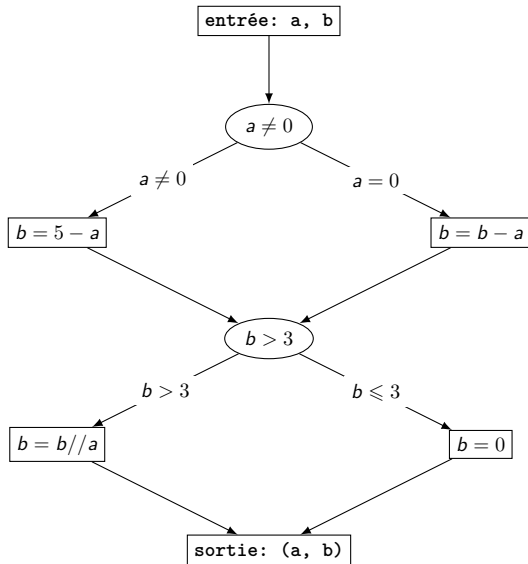
Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Pour tester `ma_fonction` il faut choisir plusieurs couples de valeurs `a`, `b`.



```
1 ma_fonction(0, 3)
2 ma_fonction(1, 3)
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

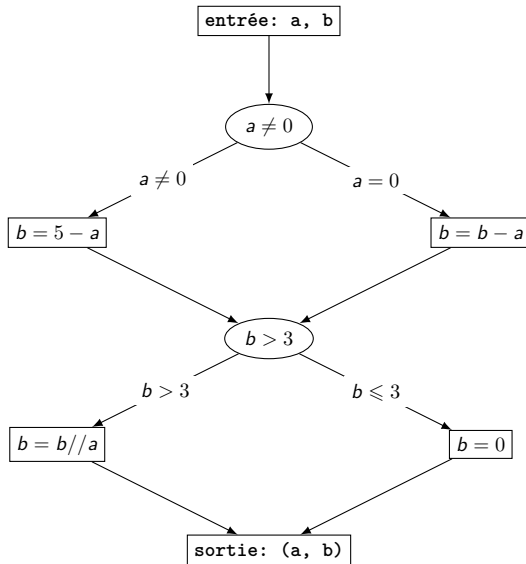
Automatiser les tests

**Tester de manière exhaustive**

## Observation

Toutes les arêtes ont été testées, cependant le risque de la division par zéro n'est pas montré.





```
1 ma_fonction(0, 4)
2 ma_fonction(2, 1)
```

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

Nécessité de tester

Repérer les cas limites

Garantir le bon comportement

Mise en évidence

Automatiser les tests

Tester de manière exhaustive

## Edsger Dijkstra

*Program testing can be used to show the presence of bugs, but never to show their absence.*

## Traduction

*Le test peut être utilisé pour démontrer la présence de problèmes, mais jamais pour démontrer leur absence.*