HEH.be Sciences et technologies

Base de programmation

- BA1 Informatique
Johan Depréter – johan.depreter@heh.be

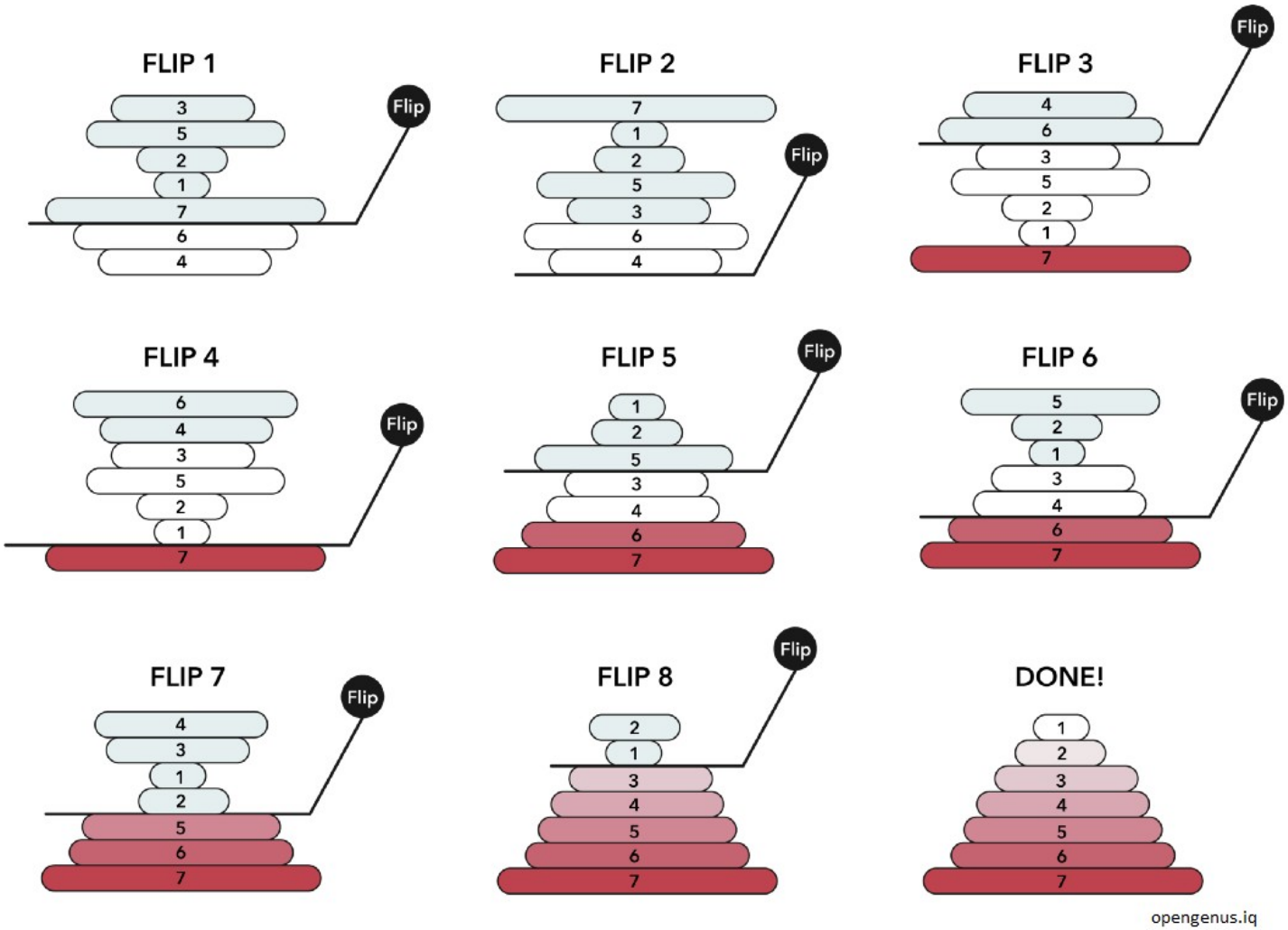
Correction

- Algorithme

  1. On va calculer la taille de la liste. Et réduire cette taille de 1 tant qu'elle est plus grande que 1. On va définir la taille en cours comme étant c

  2. Pour chaque c on va :
     a. Trouver l'index (mid) de l'élément le plus grand de la sous-liste
     b. Flip la liste jusque mid
     c. Flip la liste jusque c

Correction


FLIP 1
3
5
2
1
7
6
4
Flip
FLIP 2
7
1
2
5
3
6
4
Flip
FLIP 3
4
6
3
5
2
1
7
Flip
FLIP 4
6
4
3
5
2
1
7
Flip
FLIP 5
1
2
5
3
4
6
7
Flip
FLIP 6
5
2
1
3
4
6
7
Flip
FLIP 7
4
3
1
2
5
6
7
Flip
FLIP 8
2
1
3
4
5
6
7
Flip
DONE!
1
2
3
4
5
6
7
opengenus.iq

HEH be Sciences et technologies

Correction

```python
def flip(arr, i):
    start = 0
    while start < i:
        arr[start], arr[i] = arr[i], arr[start]
        start += 1
        i -= 1
```

```python
def findMax(arr, n):
    mid = 0
    for i in range(0, n):
        if arr[i] > arr[mid]:
            mid = i
    return mid
```

```python
def pancakeSort(arr, n):
    curr_size = n
    while curr_size > 1:
        mid = findMax(arr, curr_size)
        if mid != curr_size - 1:
            flip(arr, mid)
            flip(arr, curr_size - 1)
        curr_size -= 1
```

Implémentations

```
def bubblesort(array):
    for i in range(len(array)):
        for j in range(0, len(array) - i - 1):
            if array[j] > array[j + 1]:
                temp = array[j]
                array[j] = array[j + 1]
                array[j + 1] = temp
```

Implémentations

```
def insertsort(array):
    for i in range(1, len(array)):
        key = array[i]
        j = i-1
        while j>=0 and key < array[j]:
            array[j+1] = array[j]
            j-=1
        array[j+1] = key
```

Implémentations

```python
def mergesort(array):
    if len(array) > 1:
        mid = len(array)//2
        L = array[:mid]
        R = array[mid:]
        mergesort(L)
        mergesort(R)
        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            array[k] = R[j]
            j += 1
            k += 1
```

HEH be Sciences et technologies

Implémentations

```
def part(start, end, array):
    pivot = array[end]
    i = start-1

    for j in range(start, end):
        if array[j] <= pivot:
            i += 1
            array[i], array[j] = array[j], array[i]
    array[i+1], array[end] = array[end], array[i+1]

    return i+1

def quicksort(start, end, array):
    if (start < end):
        p = part(start, end, array)
        quicksort(start, p-1, array)
        quicksort(p+1, end, array)
```

HEH.be Sciences et technologies

Chapitre 7

Les fichiers

Introduction

- Utilisation de la mémoire vive uniquement
- Les données proviennent :
  - De l'utilisateur
  - Du code
- Besoin de stocker les données

Système de fichiers

Système de fichiers ?

- Partie du système d'exploitation responsable de la gestion de fichiers.

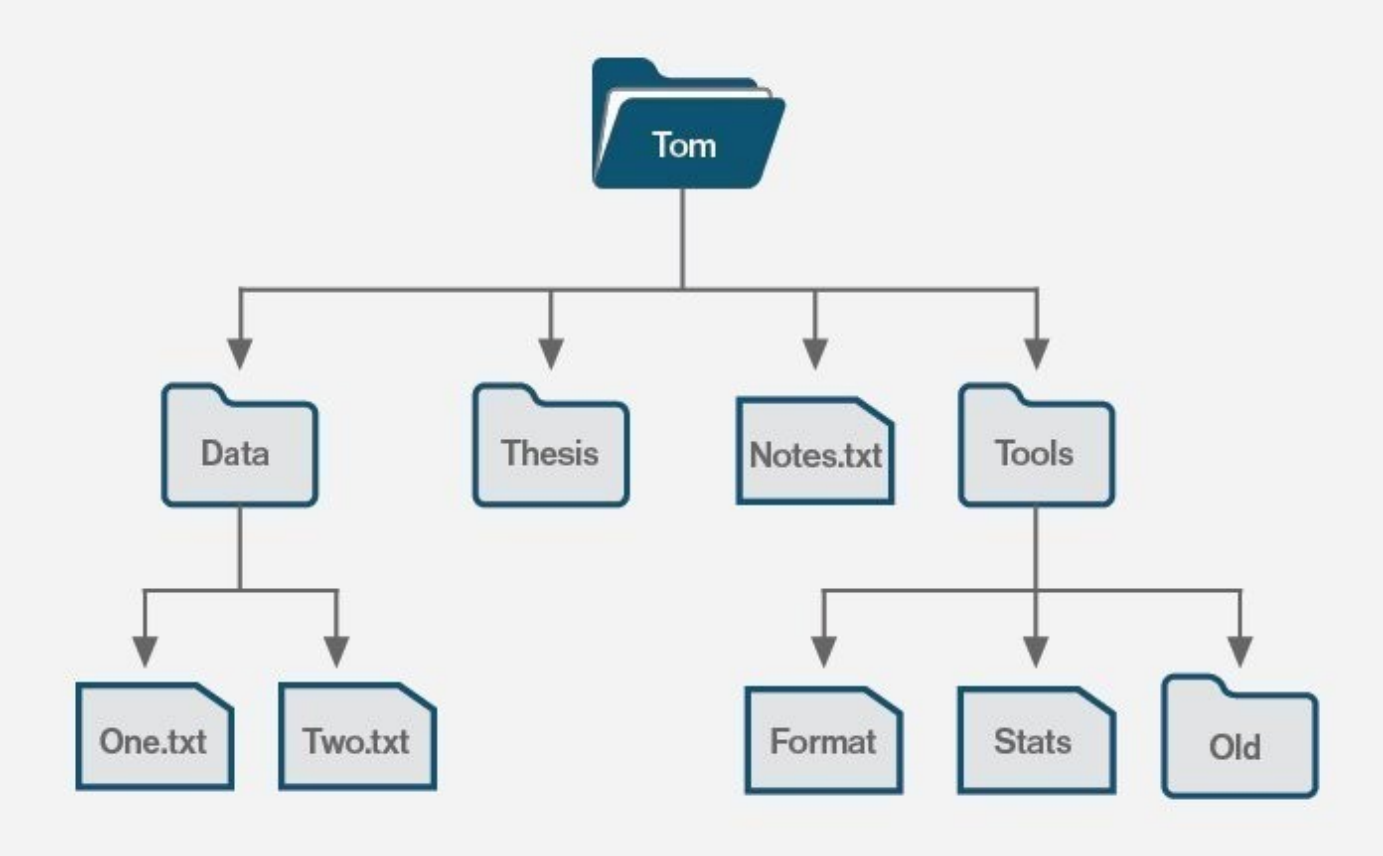
- C'est quoi un fichier ?
  - Unité de stockage logique
  - Ensemble d'informations enregistré dans le « même emplacement » mémoire.

Un fichier

- Chaque fichier a :
  - Un nom
  - Une extension
- Le nom permet de désigner le fichier avec lequel on veut travailler.
- L'extension détermine sous quel format sont les données du fichier.

Comment on y accède ?

Comment on y accède ?



Comment on y accède ?

Chemin absolu

- Exemples :

  C:\Windows\calc.exe

  /home/users/home/test.txt

- Donne le chemin complet depuis la racine

Chemin relatif

- Exemples :

  calc.exe

  ./home/test.txt

- Donne le chemin à partir de l'endroit où on se trouve

Et maintenant ?

- Flux de données

  Méthode transparente et unifiée d'envoi et réception de données

- Méthodes pour écrire dans un fichier

- Méthodes pour lire un fichier

Ouvrir un fichier

- Plusieurs modes d'ouverture :
  - 'r' – read : Permet de lire le fichier.
  - 'w' – write : Permet d'écrire dans le fichier.
  - 'a' – append : Permet d'écrire à la fin du fichier.
  - 'x' – create : Crée un fichier spécifique.
- Possibilité si on veut travailler en texte ou en binaire

Exemples

```
file1 = open("myfile.txt")

# Reading from file
print(file1.read())
```

```
file1 = open("myfile.txt" , "a" )

# Writing to file
file1.write("\nWriting to file:)" )
```

```
file = open('test.txt', 'r')

# Read the first line of the file
line = file.readline()

# Loop through the rest of the file and print each line
while line:
    print(line)
    line = file.readline()

# Close the file when you're done
file.close()
```

Stratégie d'utilisation

- IMPORTANT : Toujours fermer un fichier ouvert.

- Plusieurs méthodes :
  - Tout stocker en mémoire vive puis tout réécrire ou
  - Ecrire au fur et à mesure du programme

- On préfère la première option

Erreur

```
file1 = open("myfile.txt")

# Reading from file
print(file1.read())
```

Si myfile.txt n'existe pas

?

Erreur

Gestion des erreurs

- S’assurer que le script ne plante pas

- Passe par la gestion d’exceptions telles que :
  - TypeError
  - AttributeError
  - ValueError
  - FileNotFoundError
  - …

Gestion des erreurs

- Utilisation d'une structure de programmation « défensive »
- Try…Except (Python) Try…Catch (dans d'autres langages)
- Peut y avoir plusieurs except/catch

Gestion des erreurs

```
try:
    # Floor Division : Gives only Fractional Part as Answer
    result = x // y
    print("Yeah ! Your answer is :", result)
except ZeroDivisionError:
    print("Sorry ! You are dividing by zero ")
```

```
try:
    c = ((a+b) // (a-b))
except ZeroDivisionError:
    print ("a/b result in 0")
else:
    print (c)
```

Gestion des erreurs

- Structure générale

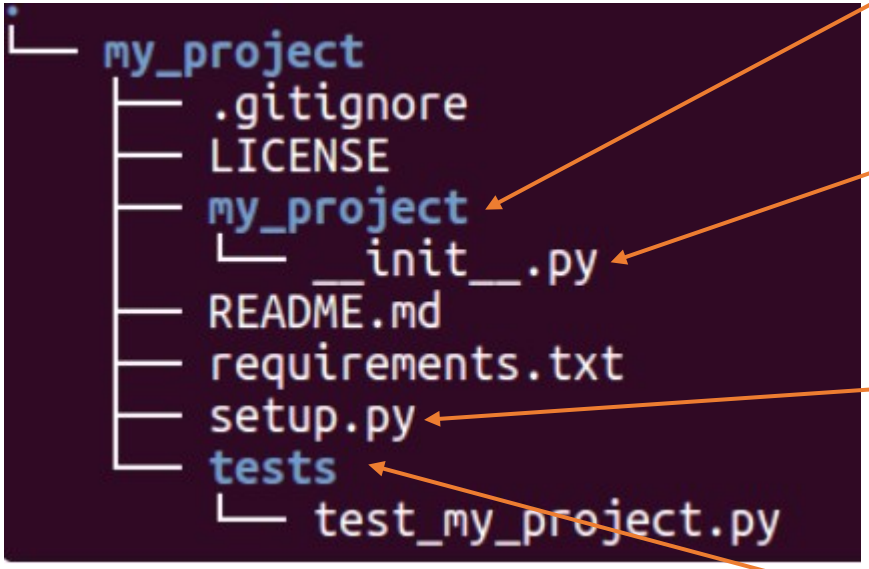
```
try:
    # Some Code
except:
    # Executed if error in the
    # try block
else:
    # execute if no exception
finally:
    # Some code .....(always executed)
```

HEH.be Sciences et technologies

Chapitre 8

Structure d’un projet et tests

Structure « classique »



- Module
- Définir que c'est un package
- Informations pour l'installation
- Tests

Module

- Contient le(s) script(s)
- Pouvoir être importé dans d’autres scripts

__init__.py

- Indique que le répertoire est un package
- Python va exécuter ce script au chargement du package

```
from os.path import dirname, abspath

ROOT_DIR = dirname(abspath(__file__))

###Logging initialisation code would go here###
```

setup.py

- Contient les informations de l'installation du package

- Exemple :

  Nom du package, paramètres d'installation, …

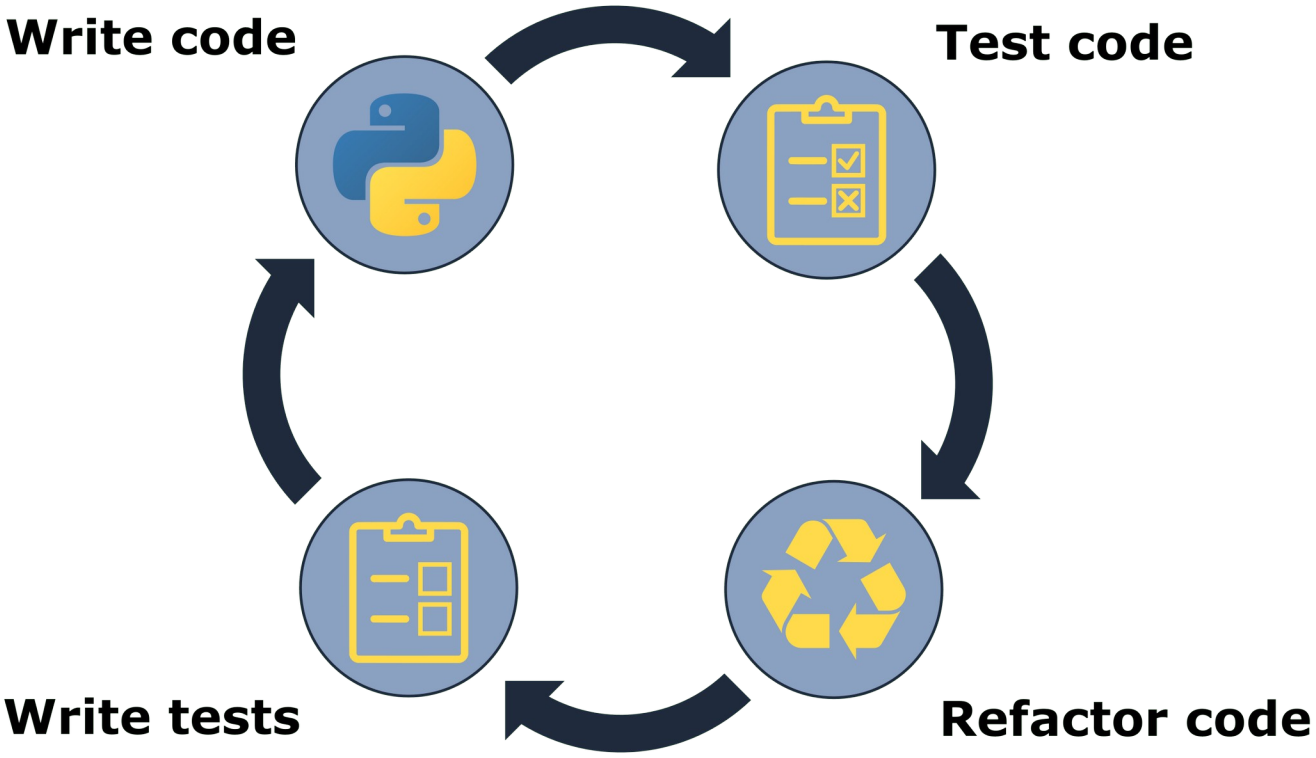
```
import setuptools


setuptools.setup(name='my_project', packages=['my_project'])
```

Tests

- Important de séparer les tests du code

- Différents types de tests :
  - Tests unitaires
  - Tests d'intégration
  - Tests de régression
  - …

Tests



Tests unitaires

- Vérifier le bon fonctionnement d'une petite partie du logiciel ou du module
- Trouver les erreurs rapidement
- « Documenter » le code

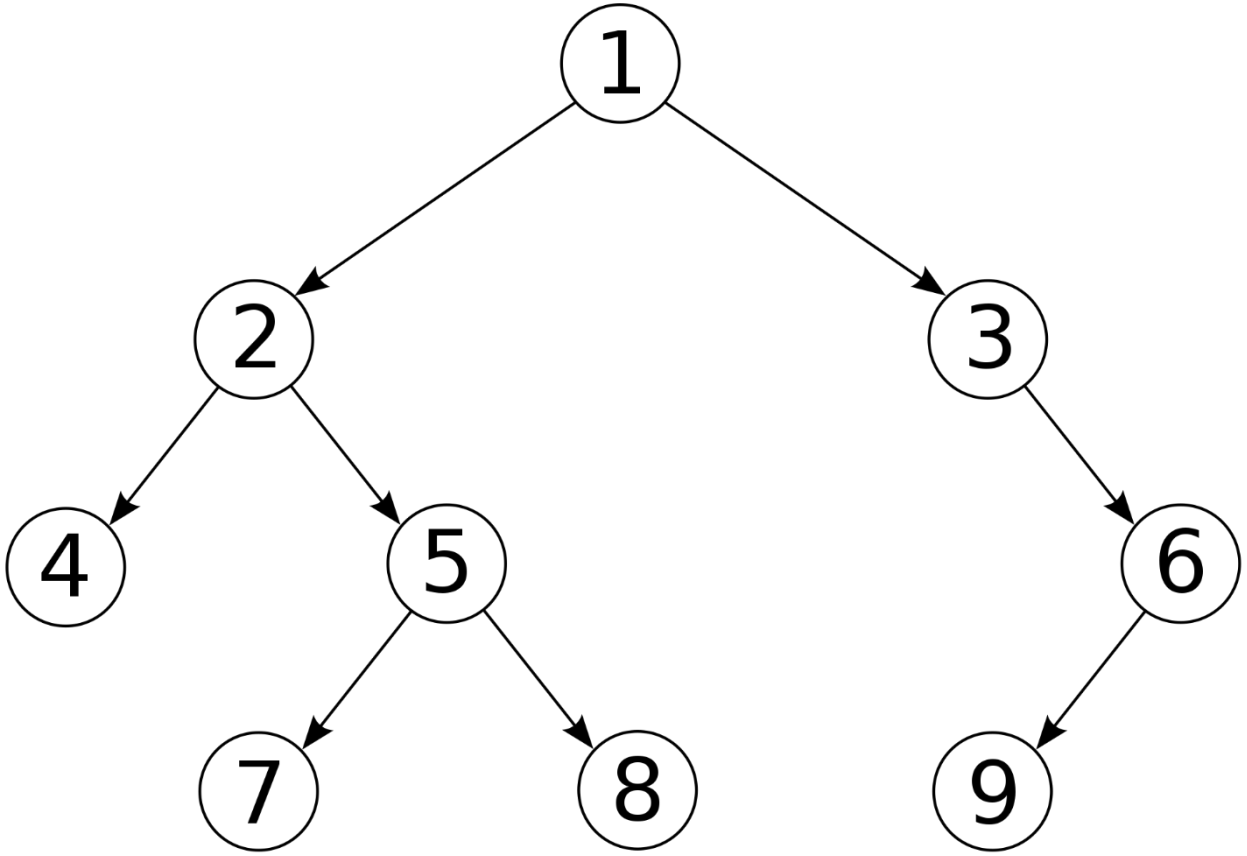
Tests unitaires

- Initialisation : définition d'un environnement de test complètement reproductible (une fixture).
- Exercice : le module à tester est exécuté.
- Vérification : comparaison des résultats obtenus avec un vecteur de résultat défini. Ces tests définissent le résultat du test : Réussi ou Echoué.
- Désactivation : désinstallation des fixtures pour retrouver l'état initial du système, dans le but de ne pas polluer les tests suivants. Tous les tests doivent être indépendants et reproductibles unitairement (quand exécutés seuls).

Tests d'intégration

- S'assurer du bon fonctionnement du programme lorsqu'on fait fonctionner les différentes fonctions ensemble
- Fonctionnement similaire aux tests unitaires
- Différentes méthodes d'approches :
  - Top-Down
  - Bottom-Up
  - Sandwich
  - Big-Bang

Tests d’intégration



Tests de régression

- Vérifier que, après modification, des bugs n'ont pas été introduits ou découverts dans des parties non modifiées du code
- Longs à exécuter
- Automatisation complexe

« Tester des programmes peut révéler des bugs très efficacement, mais cela ne permet pas d’en démontrer l’absence. »

— Edsger W. Dijkstra, *The Humble Programmer (1972)*