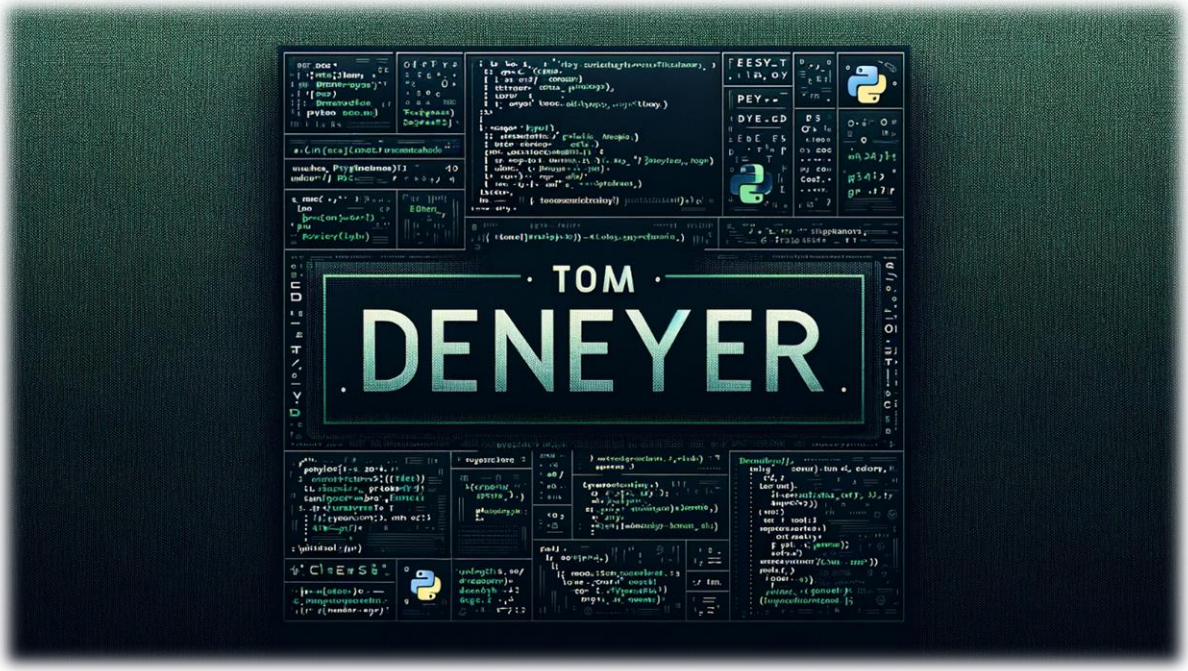
Tom Deneyer(discord: littl3t)

Programmation Théorie

Synthèse



Généralités

Rôles et propriétés physiques

- Communiquer et archiver des informations
- Traiter l'information à l'aide d'un programme

4 composants importants :

- Mémoire
- Unité de contrôle
- Unité arithmétique et logique (ALU)
- Péripheriques

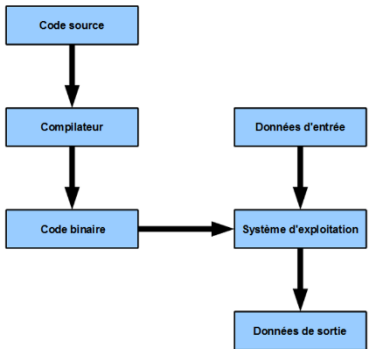
Programmes

- Ensemble d'instructions qui sont exécutés
- Programme (1) binaire = sous forme numérique, définissent un language machine

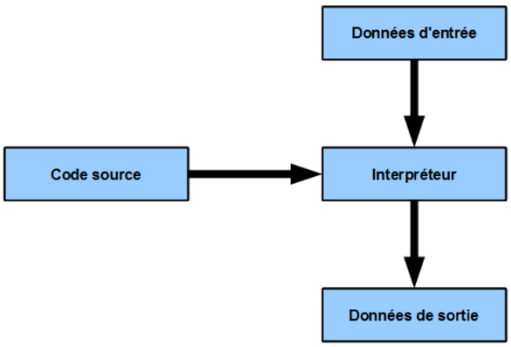
(2) source = code écrit dans un langage de programmation

Langages

- Langage le plus proche de la machine = l'assembleur
- Deux méthodes :
  - ➢
  - ➢ Compilation : (exemple C, C++)

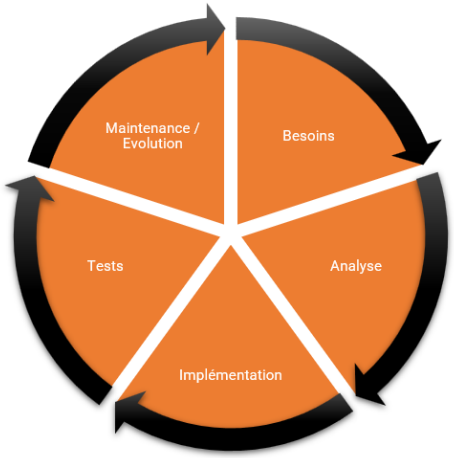


  - ➢ Interpreté: (Python, Java)

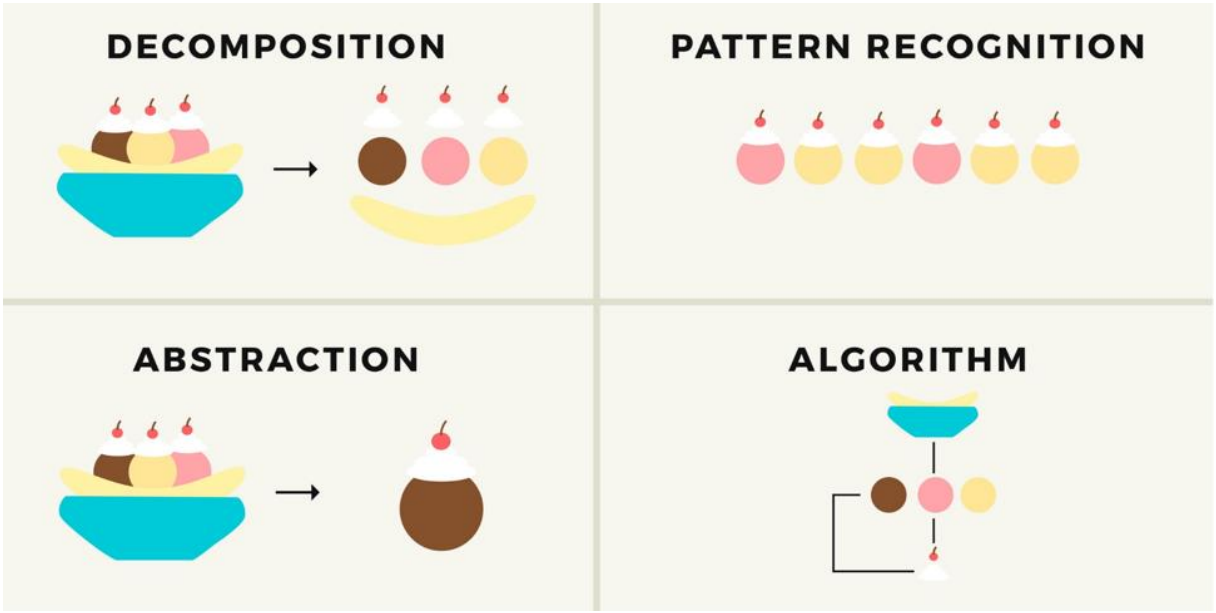


Programmation

- Désigne l'ensemble des activités permettant l'écriture des programmes informatiques
- IMPORTANT : différentes phases :
    - Définition des besoins
    - Analyse
      ➔ Déterminer les données à traiter
      + méthode et résultats
    - Implémentation
      ➔ Coder le programme
    - Tests
      ➔Unitaires, intégrations, acceptation
    - Maintenance / Evolution



Computational Thinking



Paradigmes

- Programmation impérative (procédural)
- Programmation orienté objet POO (prototype, classe)
- Programmation déclarative (fonctionnelle, logique)

Bases de la programmation

Variables

- Nom attribué à un emplacement mémoire centrale
- Conventions dépendent du langage utilisé
- Place en mémoire dépend du type…
- Types de variables :
  - Entiers (1 2 3 4 5)
  - Réels ((floats ?) 1.254 6.85 …)
  - Caractères («Felix il a un gros kiki »)
  - Booléens (1 0, True False)

Typages

- Typage Statique / dynamique
  Statique = La variable garde le même type de variable
  Dynamique = Le type de variable peut être changé au fur et à mesure (Python)
- Typage Explicite / implicite
  Explicite = Il faut déclarer le type de variable (C)
  Statique = Le type de variable est attribué automatiquement (Python)
- Typage Fort / Faible
  Fort = Ne permet pas de concaténation
  Faible = Permet une concaténation

Déclaration/affectation

- Déclaration
  Pour un language explicite, il faut avoir de la place en mémoire, permet de dire à l'ordinateur que la variable existe
- Affectation
  Placer une valeur dans la variable déclarée avant

Opérations

\+ - * / %

- Les priorité des opérations s'appliquent

\> >= < <= == !=

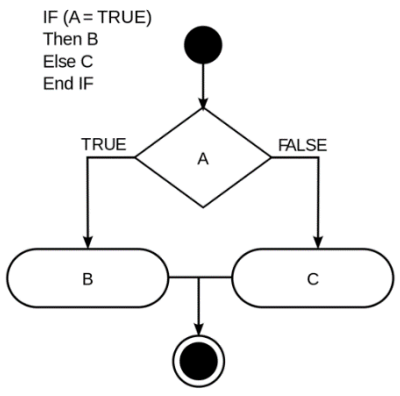
Structures conditionnelles

If

Elif

Else

(si le if est applicable, le elif n'est pas pris en compte)



Switch

```
switch(x)
{
  case 1:
            printf(" nous sommes dans le cas où x = 1");
            break;
  case 2:
            printf(" nous sommes dans le cas où x = 2");
            break;
  case 3:
            printf(" nous sommes dans le cas où x = 3");
            break;
  case 4:
            printf(" nous sommes dans le cas où x = 4");
            break;
}
```

Match

```
>>> command = 'Hello, World!'
>>> match command:
...     case 'Hello, World!':
...         print('Hello to you too!')
...     case 'Goodbye, World!':
...         print('See you later')
...     case other:
...         print('No match found')

Hello to you too!
```

Instructions Répétitives (boucles)

- Boucle avec compteur (for)
  - Prend une variable pour augmentation du compteur et s'arrête à la fin du compteur
- Boucle conditionnelle (while)
  - S'arrête quand la condition d'entrée n'est plus vraie
- Boucle de parcours (foreach)
  - Parcours les objets d'une collection, ex une liste

Spécification d'un problème

- Paramètre d'entrée
  Variable de l'énoncé
- Pré-conditions
  Condition à respecter pour que le problème aie un sens
- Paramètre de sortie
  Eléments de la réponse
- Post-conditions
  Condition à respecter par les paramètres d'entrée et de sortie (souvent méthode de résolution)

Formalisation

- Pseudo code
- Schéma

Algorithme

Définition

Méthode de résolution d'un problème de manière systématique. Même résultats dans les même conditions à chaque fois.

- Pas d'initiative
- Eviter les ambiguïtés
- Langage de programmation (clair)

Preuves

1. Terminaison
   L'algorithme se termine en un temps fini
2. La correction
   Le résultat est une solution au problème
3. La complétude
   Pour une classe de problème, l'algo donne bien l'ensemble des solutions.

Structures

➔Représenter les données efficacement…

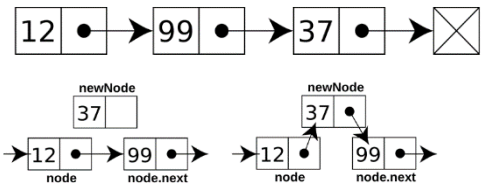
- Structures de contrôle
  - Séquence
  - Conditionnelles
  - Boucles
- Structures de données
  - Constantes
  - Variables
  - Tableaux
  - Structures récursives (listes, …)

Tableaux

- Accès par index ➔ tab[index] = x
- Données contigües
- Ajout-suppression impossible
- Attention au débordement d'indice

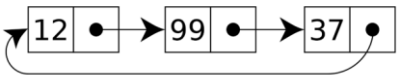
Listes chaînées

- Chaque nœud contient l'élément et un lien vers le suivant
- Pour ajouter ; créer nouveaux liens
- Pour supprimer, on modifie le lien précédent vers le suivant sans passer par l'élément à supprimer

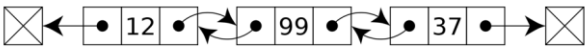


Cas particuliers

- Les listes chaînées circulaires



- Les listes doublement chaînées



Avantages / inconvénients

- Avantages
  - Données non contigües
  - Ajout/suppression après un élément ou au début très simple

- Inconvénients
  - Pas d'accès aléatoire ou d'indexation
  - Ajout/suppression avant un élément très complexe

Piles / Files

- Piles = LIFO (Last in, First out) ex: assiettes
- Files = FIFO (First In, First out) ex: feu rouge

Fonctions

Utilité:

- Répétitions de fonctionnalités
- Modularité
- Evollutivité

Fonctionnement

- Def nom_de_la_fonction(arguments_internes)
- Prend un/des paramètres ou non dans les parenthèses
- Les variables crées dans la fonction lors de sa définition restent internent (sauf si « global « variable » » est utilisé
- On appel la fonction via son nom + arguments à utiliser (nom_de_la_fonction (3))

Paramètres

- Par défaut, en donnant une valeur à l'argument lors de sa création
  Def nom_de_la_fonction(arguments_internes=0)
- Lors de l'appel on doit soit donner les arugments dans l'ordre voulu, soit donner leurs noms attribué = leurs valeurs
- Les variables interne de la fonction sont indépendantes du reste du programme

Récursivité

Fonctionnement

Une fonctionne est récursive lorsqu'elle s'appelle elle-même dans sa définition.
Il lui faut donc un cas de base, et un cas général.

Preuves

1) Preuve d'arrêt
   Bien fondé
   Appel avec des paramètres de valeurs inferieurs
2) Preuve de validité
   Correction partielle
   Démontrer que si l'ago fonctionne pour n-1 il fonctionne donc pour n

Récursif à Itératif

```python
def fibo(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibo(n - 1) + fibo(n - 2)
```


```python
def fiboI(n):
    a, b = 0, 1
    for i in range(0, n):
        a, b = b, a+b
    return a
```

Avantages / inconvénients

Récursif :

- Simple à comprendre | Donc plus intuitif
- Simple à lire ↑

- Utilisation de la mémoire ++ |Donc moins efficace
- Utilisation du CPU ++ ↑

Algorithmes de tri

Généralités

Types

- Tri en place
  ➔n'utilise pas l'élément extérieur pour trier
- Tri non en place
  ➔Utilise un élément extérieur pour trier (ex une liste tampon)
- Tri stable
  ➔L'emplacement de variable de même valeurs est connus
- Tri non stable
  ➔L'emplacement de variable de même valeurs n'est pas connus

Complexité

- Permet de mesurer la performance
- Complexité temporelle = la vitesse d'exécution
- Complexité spatiale = la mémoire utilisée
- Cela compte le nombres d'opérations nécessaire
- La taille des données est notée « n »
- Calculs dans les trois cas :
  - Meilleur cas
  - Pire cas
  - Cas moyen

Tri à Bulle

- Pour chaque élément, compare le suivant et intervertir position si nécessaire
- Complexité temporelle :

Meilleur	$O(n)$
Pire	$O(n^2)$
Moyenne	$O(n^2)$

- Complexité spatiale : O(1)
- Stable

Tri par insertion

- On parcour la liste et compare lélément avec le précédent jusqu'à trouver un élément plus grand. ➔ On compare à chaque fois avec l'élément le plus grand déjà trié. (http://lwh.free.fr/pages/algo/tri/tri_insertion.html)
- Complexité temporelle :

Meilleur	$O(n)$
Pire	$O(n^2)$
Moyenne	$O(n^2)$

- Complexité spatiale : O(1)
- Stable

Tri par fusion

Méthode récursive

- Diviser pour mieux conquérir. Diviser la liste en deux à chaque fois. Si les listes sont longue de 1 ; elle est triée. Fusionnant les listes de 1 en triant.
- Complexité temporelle :

Meilleur	$O(n \log n)$
Pire	$O(n \log n)$
Moyenne	$O(n \log n)$

- Complexité spatiale : O(n)
- Stable

Tri rapide

Tri très utilisé, principe similaire au tri par fusion

- Choix d'un pivot (ex dernier chiffre)
  Ensuite on sépare la liste en deux listes (à gauche nombres plus petit que pivot, à droite nombres plus grands que pivot + le pivot en position 0 de la liste de gauche)
  Refaire le même principe avec les deux sous listes, etc…
  Les listes de 1 de long à la fin sont donc triées, ensuite on les assemble

```python
def quicksort(liste):
    if len(liste) <= 1:
        return liste

    pivot = liste.pop()

    petit = []
    grand = []

    for nombre in liste:
        if nombre < pivot:
            petit.append(nombre)
        else:
            grand.append(nombre)

    return quicksort(petit)+[pivot]+quicksort(grand)
```

- Complexité temporelle

Meilleur	$O(n \log n)$
Pire	$O(n^2)$
Moyenne	$O(n \log n)$

- Complexité spatiale O(logn)
- Non stable

Comparatif

Nom	Meilleur cas	Pire cas	Cas moyen	Mémoire
Tri à bulles	n	n^2	n^2	1
Tri par insertion	n	n^2	n^2	1
Tri rapide	$n \log n$	n^2	$n \log n$	1
Tri par fusion	$n \log n$	$n \log n$	$n \log n$	n

Fichiers

Généralités

- Utilisation de la mémoire vive uniquement
- Système de fichiers pour stocker des données en dehors d'un programme
- Données provenant du code ou de l'utilisateur
- OS responsable de la gestion de fichiers
- Flux de données = méthode transparente et unifiée d'envoi et réception des données

Nomenclature :

Un nom + Extension

Nom = identité du fichier
Extension = Sous quel format les données sont stockées

Méthodes d'accès

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

Méthodes d'ouverture

- 'r' = read (lire le fichier)
- 'w' = write (écrire depuis le début, supprime les données déjà présentes)
- 'a' = append (écrire à la fin, ajouter aux données existantes
- 'x' = create (Crée un fichier spécifique)
- Ajouter 't' ou 'b' = texte ou binaire

Utilisation

File = open(file_name, méthode d'ouverture, encodage)

IMPORTANT ➔ Fermer fichier ouvert si pas utilisé

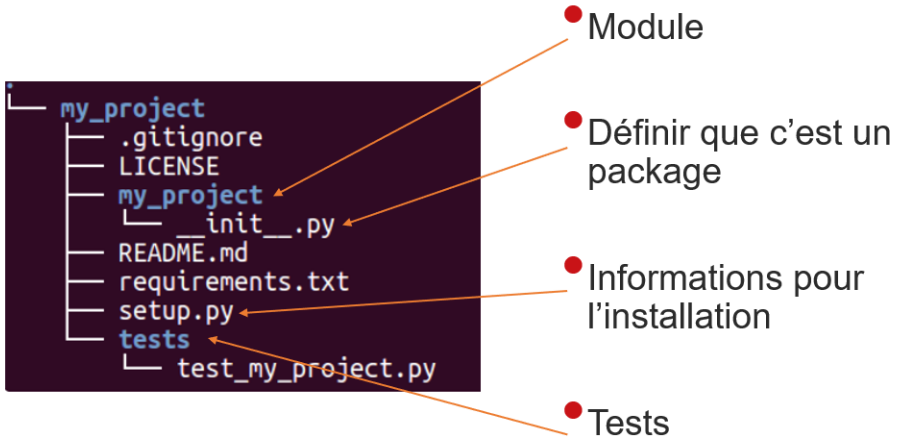
Gestion des Erreurs

Généralités

- S'assurer que le script ne plante pas
- Passe par la gestion d'exceptions connues (TypeError, AttributeError,…)
- Utilisation d'une structure défensive :
  - Try… Except(python) Try….Catch(autres langages)
- Structure de base:

```
Try:
Except : (plusieurs fois ou une fois)
Else :
Finally :
```

Structures d'un projet et Tests



Module

- Contient le(s) script(s)
- Être importé dans d'autres scripts

__Init__.py

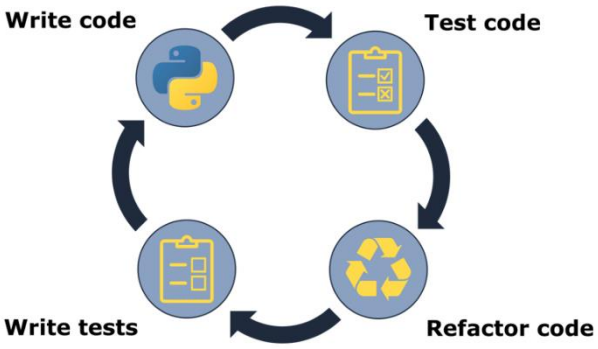
- Indique que le répertoire est un package
- Il exécute ce script au chargement du package (un peu comme un OS qui se lance au démarrage)

Setup.py

- Informations de l'installation du package :
  - Nom
  - Paramètres d'installation
  - …

Tests

- Séparer les tests du code
- Types de test :
  - Unitaire, intégration, régression, …



Test unitaire

- Vérifier une petite partie du script/module
- Trouver rapidement les erreurs
- Documenter le code

Test d'intégration

- Vérifier toutes les fonctions ensemble
- Fonctionnement similaire aux test unitaires
- Différentes méthode d'approche : Top-down, Botton-up, Sandwich, Big-Bang

Test de régression

- Vérifier après une mise à jour, qu'elle ne cause pas de nouvelles erreurs
- Long à exécuter
- Automatisation complexe

Note importante

> « 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)*

Programmation Evènementielle

Généralités

- Programme fondé sur les évènement, contrairement à la programmation séquentielle qui exécute une suite d'instruction
- L'utilisateur à le contrôle, s'il ne fait rien l'application non plus.

Interface Graphique

- Application esclave de l'utilisateur

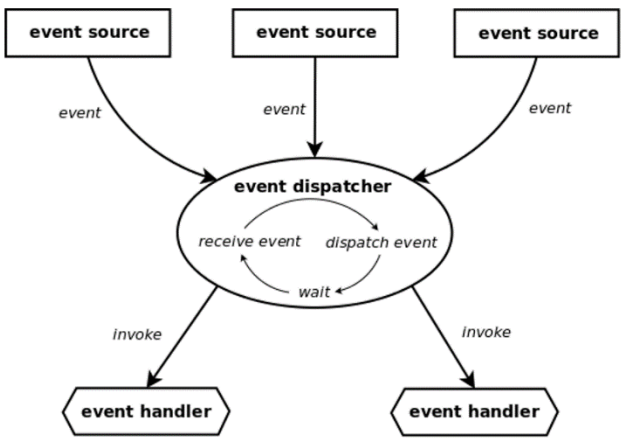
Evènements

Un évènement est un changement de l'environnement grâce à l'intervention de l'utilisateur.

2 types :

- Lié aux périphériques
  - Bouton de la souris
  - Frappe au clavier
  - Selection d'un objet dans une liste déroulante…
- Lié aux système
  - Création, Ouverture d'une fenêtre
  - Tic d'horloge
  - Mise en veille de la machine…

Fonctionnement



Compression de données

Généralités

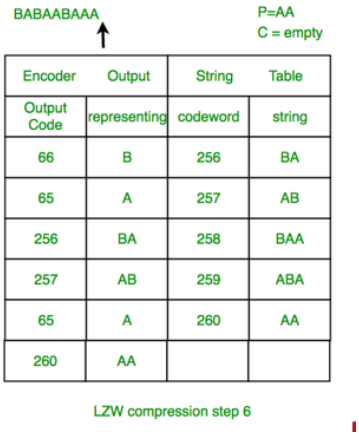
- Processus permettant de réduire le volume des données pour économiser de l'espace de stockage.
- Optimisation de l'espace important de nos jours car + en + de données

Compression Sans perte

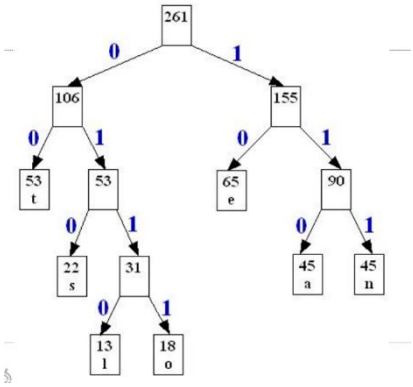
Réécriture des données sans en perdre.

- possible de décompresser les données pour revenir à l'original (exemple : txt, archive, …)
- Algo : Huffman, LZW, RLE

LWZ



1Huffman



Compression Avec perte

perte de données qui est imperceptible pour l'œil humain (ex fichier audio/vidéo)

- Les données répétitives sont remplacées par des codes plus courts.
- Les données compressées sont perdues.
- Algo : JPEG, MPEG, MP3

Résumé Python PDF

resumé python2324.pdf

https://ecampus.heh.be/pluginfile.php/219992/mod_resource/content/1/resum%C3%A9%20python2324.pdf