

Piles: Last in First out.

Est-vidé (P) : indique si la pile P est vide.

créer-pile() : crée une pile.

depiler (P) : retire le sommet

empiler (P, v) : empile la valeur v sur la pile P.

taille (P) : renvoie le nbr d'éléments dans P.

sommet (P) : renvoie le sommet de P.

def créer-pile():

return []

def depiler (P):

assert len(P) > 0

return P.pop(-1)

def taille (P):

return len(P)

def empiler (P, v):

return P.append(v)

def Sommet (P):

assert P != []

return P[-1]

def Est-vidé (P):

return P == []

fonction: inverse une pile P

def inverse (P):

P1 = créer-pile()

while len(P) > 0

x = depiler (P)

empiler (P1, x)

return P1

في نسخة Python 3

les méthodes
lus de la
déclaration d'une
nouvelle classe

File : (First in First out)

Est-vide : teste si la file est vide

creer-file : creation d'une file vide

Enfiler(f, u) : Insertion u à la fin de la file.

Defiler(f) : renvoie le 1^{er} élément (libération de la place)

taille(f) : nbr d'éléments d'une file.

Sommet(f) : élé à la tête (le 1^{er})

```
def creer-file():  
    return []
```

```
def defiler(f):  
    assert f != []  
    return f.pop(0)
```

```
def taille(f):  
    return len(f)
```

```
def enfiler(f, u):  
    return f.append(u)
```

```
def sommet(f):  
    assert f != []  
    return f[0]
```

```
def Est-vidé(f):  
    return f == []
```

```
def queue(f):  
    return f[-1]
```

Programmation orientée objets 200 :

Nom de la class
 class Domino :

objet virtuelle
 def __init__(self, a=0, b=0):

valeurs par default.
 self.faceA = a
 self.faceB = b

les attributs
 if type(a) == int / float ... and type(b) == ...

les valeurs des attributs
 else: print('error!')

def affiche_points(self):
 print('face A:', self.faceA, 'face B:', self.faceB)

def valeur(self):
 return self.faceA + self.faceB.

les methodes
 manipulation de class

objet virtuel
 خا طر باش يقيد
 كل ما تبدل

* Dans la construction __init__ :
 باحرق على les cas
 اصحاح
 خا طر باش اناقيد ولو
 ما طلبش منك.

+ نچ نزيد Parameter آخر لفة
 يبداء donnee باش نعمل بيد مقارنه
 والي نجمع بيد والي نطرح ونقسم

Class Pile :

```
Def __init__(self) :  
    self.p = []
```

```
Def bouffe(self) :  
    return len(self.p)
```

```
Def empiler(self, v) :  
    return self.p.append(v)
```

```
Def est-vide(self) :  
    return self.p == []
```

```
Def sommet(self) :  
    assert self.p != []  
    return self.p[-1]
```

```
Def depiler(self) :  
    assert self.p != []  
    return self.p.pop()
```

P = Pile()

ch = 'bonjour Bizerte'

```
Def Inverser chaine(ch, p) :
```

```
    for car in ch :  
        p.empiler(car)
```

ch2 = ""

```
    while p.est-vide() == False :
```

```
        ch2 = ch2 + p.depiler()
```

```
    return ch2
```

methode
special
= magic

→ constructeur

--init--

: appelé lors de la création d'une instance de

class par

appelée par

appelée par

nom_obj = nom classe (paramètres)

print(obj)

obj1 + obj2

- Définir une classe Pile
- Créer un objet P
- Utiliser les méthodes pour inverser les caractères d'une chaîne « bonjour Bizerte ».

← info 102!

-- repr -- (self) : appelée par >>> obj
 -- add -- (self, other)
 -- mul -- (self, other) : " obj 1 * obj 2.
 -- it -- (self, other)
 -- le --
 -- eq -- } len(self).
 -- gt --
 -- ge --

Rq: ⚠

verif_chaine (self.nom),
 fonction hors de la
 classe créée

~~verif~~
 self.nom.verif_chaine
 et inclus dans la
 classe créée

Héritage

class carre (rectangle):
 def __init__ (self, coté):

rectangle.__init__(self, coté, coté)
 ou.

super().__init__(coté, coté).

exple cercle/cylindre.

class Point:

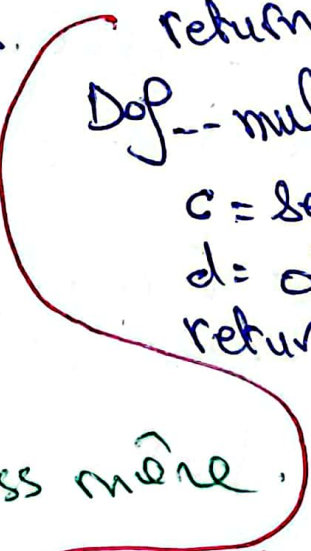
def __init__(self, a, b):
 self.x = a, (self, a=0, b=0):
 self.y = b. ↑
valeur par défaut

def __repr__(self):
 ← tuple ← return (str(self.x, self.y))

def __str__(self):
 return 'point' + str(—, —)

def __mul__(self, other):
 c = self.x
 d = other.x.
 return c * d.

'Rue:' + str(self.—) + 'ville:'
 + str(self.—) + 'code:'
 + str(self.—)



attribus point :

class Point :

def __init__ (self, x, y)

self.abs = x.

self.ord = y.

def get-abs (self):

return self.abs.

def set-abs (self, valeur)

self.abs = valeur.

affichage

↑
accesseur

modification.

↑
mutateur.

Rq : ① D.keys() → clés.

② D.values() → valeurs.

③ range (-valeur aléat

- vérification (l'entier et de type entier).

→ D[clé] = valeur

Si non

Si p. d.

Blocage

exécution.

formatage

→ manuelle : "Rue:" + self.Rue + "Gouvern:" + self.Gov

→ automatique : "Rue: %s Gouvern: %s".format (self.Rue, self.Gov).

def __eq__ (self, autre):

return self.Gov == autre.Gov.

self.num = int (input ('donner le numéro')).

+ Base de données +

Projection: $R_2 = \pi R_1 (A_1, \dots, A_p)$.

Algèbre Relationnelle

→ découpage vertical:

A ₁	A ₂	A ₃	A ₄

Selection: $R_2 = \sigma R_1 (\text{condition})$

→ découpage horizontal (n-uplets satisfaisant une condition)

Jointure: $R_3 = R_1 \bowtie R_2 (\text{condition})$.

→ Concatenation des n-uplets de R₁ et R₂ (avec condition)

Union: $R_3 = R_1 \cup R_2$.

→ Concatenation

Intersection: $R_3 = R_1 \cap R_2$.

→ les n-uplets communs.

Difference: $R_3 = R_1 - R_2$.

→ les n-uplets restants (après élimination des n-uplets communs).

Produit cartésien:

Division cartésienne.

exemple: $\pi_{A_1, A_2} R_1$
projection sur les
colonnes table
 $\pi_{A_1, A_2} (R_1)$
suffit

$\pi_{A_1, A_2} (R_1)$
(16, 10, 5)

SQL

* Création d'un tableau:

* table primaire:

CREATE TABLE nom_table (

att1 type et contrainte,

att2 "

att3 "

);

chaîne de
caractères
sans dimension.

type et contrainte → chaîne de caractères
avec nb limite.

• CHAR(n)

• TEXT

• INTEGER

• DATE

• BOOLEAN

• NOT NULL

• UNIQUE

• PRIMARY KEY

• FOREIGN KEY

• CHECK (att1 in (...))

(att2 BETWEEN
0 AND 100)

→ modification des données :

UPDATE nom table

SET liste d'affectations

(WHERE)

opérations sur les données

exple :

$prix = prix + 3$

↳ Absence de WHERE : les changements doivent être appliqués à toutes les lignes.

→ Suppression des données : (lignes).

DELETE FROM nom table.

WHERE condition

ex colonne (proportionnel)

→ extraction des données :

• SELECT att1, att2, att3
FROM nom table.

↳ sélection d'un seul tableau

≠ sélection de 2 tableaux

• SELECT t1.att1, t1.att2, t2.att1, t2.att2
FROM nomtable1 t1, nomtable2 t2.

WHERE (t1.att1 = t2.att2) exple condition.

• SELECT *
FROM nom table.

↳ affichage de toute la table.

• SELECT DISTINCT att1
FROM nom table.

↳ éliminer les doublons

• SELECT att1, att2 (avec opération).
FROM nom table.

↳ affichage avec affectation.
ex. $prix + 120$

→ Extraction avec condition.

• { SELECT att1, ~~att2~~
FROM nom_table @.

→ WHERE {
- LIKE
- BETWEEN.
- IN
- NULL
- EXISTS.

, WHERE Not (att_x = 'valeur')

• {

→ WHERE att2 = '...'. → WHERE att2 LIKE 'A%'
(le lettre A est le 2^{ème} lettre)

• {

→ WHERE att2 BETWEEN 50 AND 120
↳ 3^{ème} lettre : '...A70'
1^{ère} lettre 'A70'

• {

→ WHERE att2 IN ('...', '...', '...').

• {

→ WHERE att2 IS NULL. → WHERE cond1 AND cond2.

• {

→ ORDER BY att2. DESC ASC
ordre decroissant ordre ascendant

* SELECT t1.att1, t2.att1.

FROM t1, t2.

WHERE t1.att2 = t2.att2
ou bien.

SELECT t1.att1, t2.att1.

FROM t1 JOIN t2.

ON t1.att2 = t2.att2.

expression de
jointures.

↳ en algèbre
relationnelle.
Page 7

Rq:

WHERE : sélectionne des lignes.
AND - - - - -
AND - - - - -
plus log 20 201
au niveau des lignes.

WHERE : sélectionne des lignes.

GROUP BY : constitution des groupes. (Select Distinct)

HAVING : sélection des groupes. mais select att1, att2, att3
just distinct à att1.

SELECT (att1), att2.
FROM table1

SELECT att1, att2.
FROM table1.

GROUP BY (att1)

HAVING condition sur att1.

→ éliminer les doublons de att1.

→ Fonction de Calcul.

Count : nombre de valeurs.

Sum : Somme des valeurs.

AVG : moyenne des valeurs.

MAX : plus grande valeur

MIN : plus petite valeur.

élimination
des doublons
sur att1
jointure

* SELECT COUNT (att1) FROM table1. * SELECT sum (att2) FROM table2.

* SELECT AVG (att2) FROM table1.

SELECT t1.att1, t2.att1
FROM t1
WHERE att1 IN (select t2.att2
FROM t2)
* dans les valeurs de att2.

→ Sous-requêtes :

(=), (<), (>), (<=), (>=) ("singleton select")
(IN) (colonne unique)
(EXISTS) (jointure)

projection (jointures)

→ jointure possible.

jointure :

```
SELECT t1.att1.  
FROM table1, table2  
WHERE t1.att2 = t2.att2  
AND t2.att2 = '...'
```

→ sous requête :

```
SELECT table1 t1.att1.  
FROM table1  
WHERE t1.att1 IN
```

```
(  
SELECT t2.att2  
FROM table2  
WHERE t2.att2 = '...'
```

```
SELECT t1.att1.  
FROM table1  
WHERE NOT EXISTS ( NOT IN )
```

```
( SELECT *  
FROM table2  
WHERE t1.att2 = t2.att2 )
```

FORME générale (résumé) :

SELECT [DISTINCT].

FROM → liste de table.

WHERE → qualification

GROUP BY → attributs de partitionnement.

HAVING → qualification de groupe

ORDER BY → liste de colonnes. [ASC | DESC]

LIMIT → nbre d'enregistrements souhaités

connect() : connexion à la Base de données
cursor() : interaction avec la Base de données
execute() : évaluation d'une requête.

↳ executemany() : exécution de valeur multiples.
fetchone() : récupération d'une valeur
close() : ferme BD ↳ fetchall : "toute la valeur"
commit() : valide une transaction ≠ rollback().
input sqlite3.

ouvrir la
BD

```
conn = sqlite3.connect('nom BD')  
curseur = conn.cursor()
```

la requête

```
requete = ' ... '
```

```
curseur.execute(requete)
```

```
for ligne in curseur: fetchall()  
print(ligne) } fetchone()
```

conn.commit() → Enregist de la transaction

conn.close() → Ferme la connexion

exple requête :

```
requete = CREATE TABLE t1 ( ... )
```

```
requete = INSERT INTO t1 VALUES (?,?,?,?)
```

```
cur.execute(requete, [p1, p2, p3, p4])
```

↳ valeur retable

```
* L = curseur.execute(requete)
```

```
print(L)
```

```
requete = select att1 FROM t1 WHERE att1 = ?
```

```
cur.execute(requete, [parametre])
```

```
L = cur.fetchone()
```

↑
tuple

```
l = cur.fetchall()
```

↑
liste de tuple.

```
requete = 'CREATE - - ( . . . )'
curscur.execute(requete)
fichier = open('---.txt', "r")
```

construction de liste (mod read)

```
L = []
for ligne in fichier:
    ligne = ligne.rstrip()
    ligne = ligne.split(',')
    ligne[0] = int(ligne[0])
    L.append(ligne)
```

(supprimer '\n' en fin de la ligne)
(chaîne → liste [a1, b1, c1])
(conversion en entier) → [a1, b1, c1]

$L = [[a_1, b_1, c_1], [a_2, b_2, c_2]]$

insertion de plusieurs lignes dans une table.

```
for i in L:
    curscur.execute('INSERT INTO t1
VALUES(?, ?, ?, ?)', i)
conn.commit()
conn.close()
```

→ changement de table à partir d'un fichier texte.

2^{me} méthode: (Node a).

$L = [\text{ligne.rstrip().split(',') for ligne in fichier}]$

insertion d'une seule ligne. (cours)

selection des lignes: (cours)

→ / : division réelle. - informatique - ①
// : division entière.
% : le reste de la division entière.
** : puissance.

Int → entier.

float → réel.

str → chaîne de caractère. c = " — ".

bool → booléen. | on met l'égalité et tester le traitement (true, false)

< : inférieur

> : supérieur

<=: inf ou égal

>=: sup ou égal

== : égal

!= : différent

print(—) write.

input(—) read

structure conditionnelle: if (condition):

traitements

elif (condition):

| traitement 2

else:

traitement 3

structure répétitive:

• boucle. | for i in (éléments):
for traitement.

• structure.

• Boucle range | for i in range(5):
range print(i)
→ 0, 1, 2, 3, 4.

range(1, 10, 2)
↑ ↑ ↑
début fin (9) pas
(par défaut) par défaut
1

fonction :

(3)

def nom-fonction (liste de paramètres)

traitement 1

⋮

traitement n.

return (resultat).

def fonction() :

while true :

x = int(input("donner un entier"))

if x > 0 :

print("positif")

return(x).

* Fonction map : permet de créer une liste en appliquant
en fonction sur tout valeur itérables.

itérateur = type itérateur (map(nom-fonction, paramètre))

* fonction lambda : fonction λ sans nom

sous forme liste les
réduite.

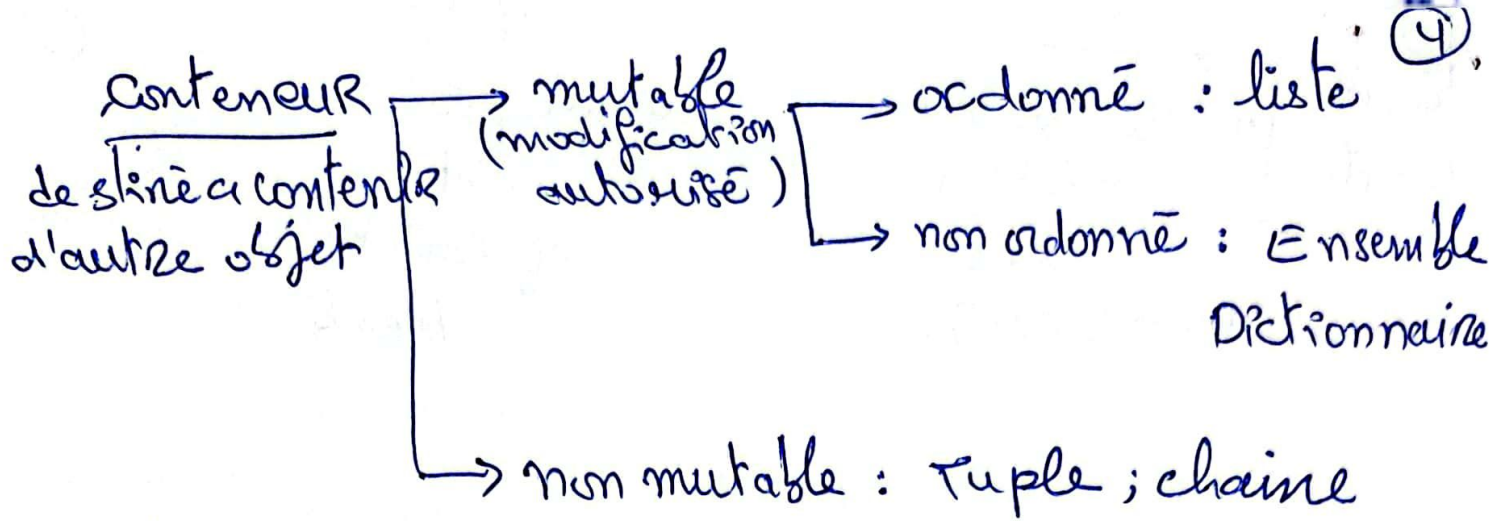
exp: somme = lambda x,y:(x+y)

⇒ somme(3,4)

7

i boucle
 while | tant que
 $x = 1$
 while $x < 7$:
 print(x)
 $x = x + 1$.
 Print("Fin").

②
repetér
 while True:
 $n = \text{int}(\text{---})$.
 if condition:
 break
 else:



liste = séquences : avec des indices (commence par 0)

$S[i]$: l'élément de l'indice i .

$S[i:j]$: tranche. $i \rightarrow j-1$

$S[i:]$: " $i \rightarrow \text{fin}$. / $S[:]$ toute la séquence

$S[i:j:k]$ " $i \rightarrow j-1$, pas k .

$\text{len}(S)$: longueur

$\text{max}(S)$; $\text{min}(S)$: plus grand, petit élément dans S .

$S.\text{index}(i)$: indice i $\rightarrow$ i هو i هو

$S.\text{count}(i)$: nombre i $\rightarrow$ i هو

Chaîne :

$ch = \text{'python'}$. $ch[0] \rightarrow \text{'p'}$ $ch[-1] \rightarrow \text{'n'}$

$ch_2 = ch + ch_1$; $ch_1 = \text{'cours'}$.

$\Rightarrow ch \rightarrow$
python cours.

$\rightarrow$ chaîne على $\rightarrow$ $\rightarrow$

non mutable : $\rightarrow$ $\rightarrow$

حاجتي به $\rightarrow$ $\rightarrow$

• Extra - t. d'oppar lue - Répétition - convertir

• `ch.find('lettre')` : l'indice de lettre

`sinon` $\rightarrow$ (-2) : n'existe pas.

• `ch.upper()` : en majuscule.

• `ch.lower()` : convertir en minuscule.

• `ch.title()` : مِنْ تَوَاجُهَات ; المَاجِيسْكُول
majuscule.

• `ch.capitalize()` : مِنْ تَوَاجُهَات ; المَاجِيسْكُول

• `ch.replace('lettre', '_')` $\rightarrow$ تَوَاجُهَات.

• strip : سَافَاة
سَافَاة

وَأَتَا حَرْفَات
فَرَسَات فِي بَاق

`ord(caractère)` $\rightarrow$ code
asciè
`chr('codeascii')` $\rightarrow$ caractère

liste : [1, 2, 3, K]
voir cours détail.

chaîne $\rightarrow$ liste.

`L = ch.split()`

tuple : ('a', 1, 'c', 27)

voir cours détail;

chaîne $\rightarrow$ tuple : `t = tuple(ch)`

liste $\rightarrow$ tuple : `t = tuple(L)`.

creation d'une matrice :

```
import numpy as np
import random as rd
```

input random as $n \rightarrow \infty$

$$M = np.zeros((n, m), dtype = int)$$

amplification
terme à
terme.

for i in range n :

for j in range m :

$$\pi[x, y] = \dots$$

simplification
avec
valeurs entières
aléatoires.

for i in range n :

for j in range m :

$$m[i, j] = \text{nd. gradient}(0, \forall f).$$
$$(0, 1, \dots, \nu_0 - 1).$$

same impact.

$$r[i, j] = \text{rand} \cdot \text{randint}(0, \text{up})$$

taille de Matrice :

$$r, m = r, \text{shape}$$

on

$$\begin{cases} \text{or} \\ n, m = nq \cdot \text{shape}(\pi) \end{cases}$$

copier les lignes ou colonnes:

$V[0, :] = M[0, :]$ toute la colonne.

$$V[0,0:m] = M[0,0:m] \text{ partic.}$$
$$v = \pi \cdot \log y() \quad ; \quad \frac{1}{\log y}$$

$a_1 = \text{np.array}([4, 7, 9])$.
 $\rightarrow \text{array}([4, 7, 9])$.

tableau
monodimensionnel

type(a) :

$\rightarrow \text{numpy.ndarray}$

$a_2 = \text{np.array}([1, 2, 3], [4, 5, 6])$.

$\rightarrow ([[1, 2, 3], [4, 5, 6]])$. matrice (Tableau)
Bidimensionnelle.

$\text{np.size}(a_1)$ $\rightarrow 3$	{	$\text{np.shape}(a_1)$ $\rightarrow (4,)$
$\text{np.size}(a_2)$ $\rightarrow 6$		$\text{np.shape}(a_2)$ $\rightarrow (2, 3)$

produit : $a_1 \times a_2$.
~~tenue~~
~~ta~~

produit matriciel :

colonne de a = ligne de b.

$\text{np.dot}(a, b)$.

transposé :

$\text{cross.}()$

$\text{np.transpose}(a)$.

determinant :

$\text{alg.det}(A)$.

trace :

$\text{np.trace}(A)$.

inverse :

$\text{alg.inv}(A)$

rang :

$\text{alg.matrix_rank}(A)$.

→ `diag (A, 0)`.

matrice par bloc :

`np.concatenate ((A, B), axis=0)`.

← تلف على بعض

matrice diagonal :

`np.diag ([1, 2, 3])`.

→ $\begin{pmatrix} [1, 0, 0], \\ [0, 2, 0], \\ [0, 0, 3] \end{pmatrix}$.

slicing

`np.cross (a, b)`,
produit vectoriel.

`a = array ([12, 25, 34, 56, 87])`.

→ `a[1:0]` ⇒ `[25, 34, 56, 87]`.

→ `a[:3]` ⇒ `[12, 25, 34]`.

→ `a[:]` ⇒ `[12, 25, 34, 56, 87]`.

`a = np.array ([[1, 2, 3],
[4, 5, 6]])`.

ligne

colonne

→ `a[0, 1]` ⇒ 2.

→ `a[:, 1:2]`

⇒ $\begin{bmatrix} [2], \\ [5] \end{bmatrix}$.

→ `a[:, 1:3]` ⇒ $\begin{bmatrix} [2, 3], \\ [5, 6] \end{bmatrix}$.

→ `a[:1]` ⇒ `[2, 5]`.

→ `a[0, :]` ⇒ `[1, 2, 3]`.

calcul de somme :

$$L = [\pi(a, b) \text{ for } a \text{ in range}(i-1, i+1+1) \\ \text{for } b \text{ in range}(j-1, j+1+1)] \\ S = \text{sum}(L).$$

Convertir image en Matrice :

$M = \text{plt.imread}('nom.jpg')$

$M = \text{mpimg.imread}('nom.jpg')$.

Convertir Matrice en Image :

$\text{image} = \text{plt.imshow}(M)$ (Image est tableau).

$\text{plt.show}()$ (image).

Creation d'une matrice vide. $\left. \begin{array}{l} n \text{ lignes} \\ m \text{ colonnes} \\ \text{entiers} \end{array} \right\}$

$M = \text{np.ndarray}(\text{shape}=(n,m), \text{dtype}=\text{int})$.

$M = \text{np.zeros}((n,m), \text{dtype}=\text{int})$.

$M = \text{np.empty}((n,m), \text{dtype}=\text{int})$.

taille de la matrice M :

$(n,m) = r.\text{shape}$.

$(n,m) = \text{np.shape}(M)$.

selection d'une sous-image :

$B_2 = B_1[0:100, 0:100]$.

$\text{im} = \text{plt.imshow}(B_2)$.

$\text{im.set_cmag}('gray')$.

$\text{plt.show}()$.

Intensité de l'image :

$I = B_1 * B_1$ ← plus intense : 0.255

$D = np.concatenate([B_1, I], axis=0)$

$im = plt.imshow(D)$

$im.set_cmap('gray')$

$plt.show()$

②

$C = B_1 * B_1$

$E = np.concatenate([B_1, C], axis=1)$

$im = plt.imshow(E)$

$im.set_cmap('gray')$

$plt.show()$

* Image de départ (noise).

$image = np.zeros((hauteur, largeur, 3))$

* Remplissage :

$image[:, 0:largeur // 3] = bleu$

$image[:, largeur // 3:2 * largeur // 3] = violet$

$image[:, 2 * largeur // 3:largeur] = rouge$

$plt.imshow(image)$

$plt.show()$

$plt.savefig("image2.png", image)$

* Symétrie / axe vertical du milieu :

Def symetrie(im):

$n, p, r = im.shape$

$im_sym = np.zeros((n, p, r))$

for i in range(n):

for j in range(p):

$im_sym[i, j] = im[i, p-1-j]$

return im_sym

* Seillage : Pixel = 0 pour celle (fixe)
 def seillage(im):
 n, p, s = im.shape
 for i in range(n):
 for j in range(p):
 for k in range(s):
 if im[i, j, k] < 150 :
 im[i, j, k] = 0.
 return

import matplotlib.pyplot as plt
 import matplotlib.image as mpimg
 import numpy as np.

b = [- - - -]
 image = plt.imshow(b).
 plt.show.

tableau → image
 (image → tableau)
 (im.imread)

img = plt.imshow('gray').
 plt.show

نور بياض

mat = np.zeros((n, m)).

mat[:, 0, : - - -] = couleur

image = plt.imshow(mat)
 plt.show

[a, b, g]
 pixel.

noirs
 ↓
 bleu
 —————
 image
 à 3 couleurs