I. Introduction à Python

➔ Expressions de base : Voir datasheets

➔ TP : voir le fichier « python_tutorial.ipynb »

II. Introduction aux modules NumPy et Matplotlib

En Python, certains modules permettent de faciliter la manipulation de données numériques et leur représentation graphique. Les deux modules les plus utilisés dans ce domaine sont NumPy et Matplotlib. Ils sont très utilisés en science des données, en traitement d'image, en intelligence artificielle, et dans de nombreux autres domaines scientifiques.

Le module NumPy

NumPy (Numerical Python) est une bibliothèque qui permet de manipuler des tableaux de nombres (appelés ndarray) de façon efficace.

Les images peuvent être facilement manipulées en Python grâce aux tableaux NumPy.

Importation du module : import numpy as np

On utilise souvent l'alias np pour raccourcir l'écriture.

Création d'un tableau

```
import numpy as np

A = np.array([1, 2, 3])

B = np.array([[1, 2], [3, 4]])
```

A est un tableau 1D (une ligne). B est un tableau 2D (matrice).

- A.shape # Renvoie la taille du tableau
- A.dtype # Type des éléments (int, float, etc.)
- A.ndim # Nombre de dimensions
- A.size # Nombre total d'éléments

Le module Matplotlib

Matplotlib est une bibliothèque pour **tracer des courbes, des graphiques** et visualiser des données.

Importation du module : import matplotlib.pyplot as plt

Tracer une courbe simple

```
import numpy as np

import matplotlib.pyplot as plt

# Création des données

x = np.linspace(0, 10, 100)  # 100 points entre 0 et 10

y = np.sin(x)              # Calcul du sinus de chaque point

# Création du graphique

plt.figure(figsize=(8, 4))   # Taille de la figure (largeur, hauteur) en pouces

plt.plot(x, y)             # Tracer la courbe

plt.title("Fonction sinus")  # Titre du graphique

plt.xlabel("x")            # Étiquette de l'axe x

plt.ylabel("sin(x)")       # Étiquette de l'axe y

plt.grid(True)             # Afficher une grille

plt.savefig("sinus.png")     # Sauvegarder l'image (optionnel)

plt.show()                 # Afficher le graphique
```

Personnalisation des courbes

```
# Plusieurs courbes sur le même graphique

plt.figure(figsize=(10, 6)) # Crée une nouvelle figure Matplotlib avec une taille de 10 pouces de largeur et 6 pouces de hauteur

plt.plot(x, np.sin(x), label="sin(x)", color="blue", linestyle="-", linewidth=2)
```

```
plt.plot(x, np.cos(x), label="cos(x)", color="red", linestyle="--", linewidth=2)

plt.legend()  # Afficher la légende

plt.show()
```

Types de graphiques

Matplotlib propose de nombreux types de graphiques :

Nuage de points (Scatter plot)

```
x = np.random.rand(50) # Génère 50 nombres aléatoires uniformément répartis entre 0 et 1 pour l'axe X
y = np.random.rand(50) # Génère 50 nombres aléatoires uniformément répartis entre 0 et 1 pour l'axe y
plt.figure(figsize=(8, 6))

plt.scatter(x, y, color="green", marker="o", s=100, alpha=0.5) # Crée un nuage de points avec :
# - x, y : coordonnées des points
# - color="green" : couleur des points (vert)
# - marker="o" : forme des points (cercle)
# - s=100 : taille des points
# - alpha=0.5 : transparence des points (50% transparent)
plt.title("Nuage de points")

plt.grid(True)  # l'affichage de la grille
plt.show()   # Affiche la figure à l'écran
```

Histogramme

```
data = np.random.normal(0, 1, 1000)  # 1000 points suivant une loi normale

plt.figure(figsize=(8, 6))

plt.hist(data, bins=30, color="purple", alpha=0.7, edgecolor="black") # Crée un histogramme avec :
# - data : données à représenter
# - bins=30 : nombre de barres/intervalles (30 barres)
# - color="purple" : couleur de remplissage des barres (violet)
# - alpha=0.7 : transparence des barres (70% opaque)
# - edgecolor="black" : couleur des bordures des barres (noir)
```

```
plt.title("Histogramme")

plt.xlabel("Valeur") # Ajoute une étiquette à l'axe des abscisses (axe X)

plt.ylabel("Fréquence") # Ajoute une étiquette à l'axe des ordonnées (axe Y)

plt.grid(True)

plt.show()
```

III. Affichage et génération d'image

Les images peuvent être facilement manipulées en Python grâce aux tableaux NumPy.

```
import numpy as np

import matplotlib.pyplot as plt
```

Chargement et affichage d'images réelles

Pour travailler avec des images réelles, on peut utiliser la fonction **plt.imread** de la bibliothèque matplotlib pour les charger sous forme de tableaux NumPy.

```
import numpy as np

import matplotlib.pyplot as plt

# Charger l'image avec imread

img_array = plt.imread(/chemin/image.png') # Charge l'image 'image.png' dans un tableau numpy

# Afficher les dimensions

print(img_array.shape)  # Affiche les dimensions du tableau (nombre de pixels en hauteur, nombre de pixels en largeur, nombre de canaux)

# Afficher l'image

plt.imshow(img_array)  # Affiche l'image chargée

plt.title("Image originale")

plt.axis('off')
```

```
plt.show()
```

IV. Image binaire

```
# Création d'une image binaire 10x10

binary_image = np.random.randint(0, 2, size=(10, 10)) # Génère une matrice 10x10 remplie de 0 et 1 aléatoires


plt.figure(figsize=(4, 4)) # Crée une figure de 4x4 pouces

plt.imshow(binary_image, cmap='binary') # Affiche l'image binaire avec une colormap binaire (noir et blanc)


plt.title("Image binaire")

plt.axis('off')

plt.show()
```

V. Image en niveaux de gris

```
# Création d'une image en niveaux de gris 8x8

grayscale_image = np.zeros((8, 8), dtype=np.uint8) # Crée une matrice 8x8 remplie de zéros (noir) de type entier non signé 8 bits

for i in range(8): # Boucle sur les 8 lignes de l'image de 0 à 7

    grayscale_image[i, :] = i * 32  # Dégradé horizontal : chaque ligne a une intensité croissante (0 à 224 par pas de 32)

plt.figure(figsize=(4, 4))

plt.imshow(grayscale_image, cmap='gray', vmin=0, vmax=255) # Affiche l'image en niveaux de gris avec la colormap 'gray'

plt.title("Image en niveaux de gris")
```

```
plt.axis('off')

plt.show()
```

VI. Image couleurs

Espaces couleur

Image en couleur (RGB)

```
# Création d'une image couleur 8x8x3

color_image = np.zeros((8, 8, 3), dtype=np.uint8) # Crée un tableau 8x8x3 (3 canaux pour RGB) rempli de zéros

color_image[0:4, 0:4] = [255, 0, 0]     # Rouge   # Quadrant supérieur gauche : Rouge (R=255, G=0, B=0)

color_image[0:4, 4:8] = [0, 255, 0]     # Vert # Quadrant supérieur droit : Vert (R=0, G=255, B=0)

color_image[4:8, 0:4] = [0, 0, 255]     # Bleu # Quadrant inférieur gauche : Bleu (R=0, G=0, B=255)

color_image[4:8, 4:8] = [255, 255, 0]     # Jaune # Quadrant inférieur droit : Jaune (R=255, G=255, B=0)

color_image[2, 5] = [255, 128, 0] # Pixel à la ligne 2, colonne 5. Rouge = 255, Vert = 128, Bleu = 0

plt.figure(figsize=(4, 4))

plt.imshow(color_image)

plt.title("Image en couleur (RGB)")

plt.axis('off')

plt.show()
```

Création et affichage de l'image numérique A + le négatif

Skimage est une collection d'algorithmes pour le traitement d'image et vision par ordinateur. **scikit-image** est un paquet open source Python qui fonctionne avec les tableaux numpy et qui est utilisé pour le traitement des images.

```
import numpy as np

import matplotlib.pyplot as plt

from skimage import io #importe le module io (Input/Output) de la bibliothèque scikit-image. C'est le module qui gère la lecture, l'écriture et l'affichage des images.

# image A : dessin géométrique

dessin=255*np.ones((300,200,3),dtype=np.uint8) # création d'une image blanche de dimensions 300x200 avec 3 canaux (RGB) # 255*np.ones() : initialise tous les pixels à blanc (255,255,255)

dessin[100:120,:,:]=0 # bande noire horizontale # Modifie les lignes 100 à 119 (exclue 120) sur toute la largeur. [100:120, :, :] = 0 : met tous les canaux (R,G,B) à 0 → noir

dessin[10:90,10:60,1:]=0 # rectangle rouge à gauche

# rectangle orange à droite

dessin[10:90,140:190,1],dessin[10:90,140:190,2]=120,0 #  R=255, G=120, B=0 → orange # ou écrire :

dessin[10:90,140:190,:]=(255,120,0)

dessin[10:90,70:130,2]=0 # rectangle jaune citron au centre

dessin[130:270,70:130,0],dessin[130:270,70:130,2]=0,0 # rectangle vert au centre

dessin[190:240,10:60,:2],dessin[190:240,140:190,:2]=0,0 # carrés bleus à gauche et à droite

io.imshow(dessin)

#plt.show() nécessaire pour certaines versions de Python
```

```
#négatif du dessin : Le négatif s'obtient en soustrayant chaque valeur de pixel de 255
# 255 - dessin : inverse toutes les couleurs
# Blanc (255,255,255) → Noir (0,0,0)
# Noir (0,0,0) → Blanc (255,255,255)
# Couleurs intermédiaires : leur complémentaire
io.imshow(255-dessin)
```

Modifications d'image en nuances de gris

```
# fonction pour obtenir un tableau représentant l'image en niveau de gris (version boucle)

def gris(T): # Récupère les dimensions de l'image T : lx = nombre de lignes (hauteur), ly = nombre de colonnes (largeur), lz = nombre de canaux (3 pour RGB)
    lx, ly, lz = T.shape

    M = np.zeros((lx, ly))  # Crée un tableau vide de dimensions (lx, ly) rempli de zéros pour stocker l'image en niveaux de gris


    N = np.array(T, dtype=np.float32) # Convertit l'image T en tableau NumPy de type float32 pour permettre des calculs avec des décimales
    for i in range(lx): # Boucle sur chaque pixel de l'image (parcours ligne par ligne)
        for k in range(ly): # Boucle sur chaque colonne de la ligne i
            M[i, k] = 0.2125 * N[i, k, 0] + 0.7154 * N[i, k, 1] + 0.0721 * N[i, k, 2] # Calcule la valeur de gris du pixel (i,k) en combinant les canaux R, G, B # Coefficients standards pour la perception humaine : 21.25% rouge, 71.54% vert, 7.21% bleu
    return np.clip(M, 0, 255).astype(np.uint8) # Limite les valeurs entre 0 et 255 (au cas où les calculs dépasseraient cette plage)# puis convertit en type uint8 (entiers non signés 8 bits) standard pour les images


B = gris(dessin) # Appelle la fonction gris() avec l'image 'dessin' comme paramètre # Le résultat B est une version en niveaux de gris de l'image originale
plt.subplot(2, 2, 3) # Crée une sous-figure à la position (2,2,3) dans une grille 2x2 (3ème position)


plt.title("Niveaux de gris (version boucle)")
```

```
io.imshow(B, cmap='gray') # cmap='gray' spécifie d'utiliser une palette de gris pour l'affichage
# autre fonction utilisant le calcul vectoriel (plus efficace)

def gris2(T):
    N = np.array(T, dtype=np.float32)  # Convertit l'image T en tableau NumPy de type float32 pour les calculs
    M = 0.2125 * N[:, :, 0] + 0.7154 * N[:, :, 1] + 0.0721 * N[:, :, 2] # Conversion selon les coefficients de luminance standard
    # Version vectorisée : calcule directement sur tous les pixels sans boucle
    # N[:, :, 0] = tous les pixels, tous les canaux rouge (indice 0)
    # N[:, :, 1] = tous les pixels, tous les canaux vert (indice 1)
    # N[:, :, 2] = tous les pixels, tous les canaux bleu (indice 2)
    return np.clip(M, 0, 255).astype(np.uint8)  # conversion en entiers uint8

B2 = gris2(dessin)  # B2 est l'image en niveau de gris (version vectorisée) calculée plus rapidement


plt.subplot(2, 2, 4) # Crée une sous-figure à la position (2,2,4) dans une grille 2x2 (4ème position)
plt.title("Niveaux de gris (version vectorisée)")

io.imshow(B2, cmap='gray') # Affiche l'image en niveaux de gris B2 avec palette de gris


plt.tight_layout()# Ajuste automatiquement l'espacement entre les sous-figures pour éviter les chevauchements


plt.show()# Affiche toutes les figures créées (les 4 sous-figures)


# Vérification que les deux méthodes donnent le même résultat

print(f"Les deux méthodes donnent-elles le même résultat ? {np.array_equal(B, B2)}") # np.array_equal() compare élément par élément si les deux tableaux sont identiques # Affiche True si B et B2 sont identiques, False sinon
# Affichage des informations sur les images

# \n : Affiche une ligne vide pour séparer visuellement les résultats
```

```
print(f"\nDimensions de l'image originale : {dessin.shape}") # Affiche les dimensions de l'image originale (hauteur, largeur, canaux)


print(f"Dimensions de l'image en niveaux de gris : {B.shape}") # Affiche les dimensions de l'image en niveaux de gris (hauteur, largeur) - seulement 2 dimensions


print(f"Valeurs min/max dans l'image originale : {dessin.min()}/{dessin.max()}")# Affiche les valeurs minimale et maximale des pixels dans l'image originale (devrait être 0/255)


print(f"Valeurs min/max dans l'image en niveaux de gris : {B.min()}/{B.max()}")# Affiche les valeurs minimale et maximale dans l'image en niveaux de gris
```

```
# Jaune pur (255, 255, 0) :
niveau_gris = 0.2125*255 + 0.7154*255 + 0.0721*0
niveau_gris = 54.19 + 182.43 + 0 = 236.62 ≈ 237

# Bleu pur (0, 0, 255) :
niveau_gris = 0.2125*0 + 0.7154*0 + 0.0721*255
niveau_gris = 0 + 0 + 18.39 = 18.39 ≈ 18

# C'est pourquoi un bleu pur apparaît très foncé en niveaux de gris.
```

VII. Manipulation d'histogramme

Un histogramme d'image est une représentation graphique de la distribution des niveaux de gris ou des intensités de couleur d'une image. L'axe horizontal représente les valeurs d'intensité, tandis que l'axe vertical représente le nombre de pixels ayant cette intensité. Les histogrammes sont couramment utilisés pour évaluer la luminosité, le contraste et la qualité d'une image.

Objectif de l'exercice : Calculer et afficher l'histogramme d'une image.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color
```

```
from skimage.exposure import histogram as
sk_histogram # importer la function histogram du
module skimage.exposure et le renommer sk_histogram

# Charger l'image avec skimage
image = io.imread('/content/image.jpg')

# Convertir en niveaux de gris si l'image est en
couleur
if len(image.shape) == 3:
    image = color.rgb2gray(image)
    # Convertir en uint8 pour avoir des valeurs entre
0-255
    image = (image * 255).astype(np.uint8)

# Calculer l'histogramme avec skimage
hist, bin_centers = sk_histogram(image, nbins=256)

# Alternative : utiliser numpy pour calculer
l'histogramme
# hist, bins = np.histogram(image.ravel(), bins=256,
range=(0, 256))

# Afficher l'histogramme
plt.figure(figsize=(10, 5))

# Méthode 1 : Utiliser matplotlib directement
plt.hist(image.ravel(), bins=256, range=(0, 256),
alpha=0.7, color='blue', edgecolor='black')
plt.title('Histogramme de l\'image avec skimage')
plt.xlabel('Niveau d\'intensité')
plt.ylabel('Fréquence')

# Méthode 2 : Afficher l'histogramme calculé par
skimage
# plt.bar(bin_centers, hist, width=1, alpha=0.7,
color='blue', edgecolor='black')
# plt.title('Histogramme de l\'image avec skimage')
# plt.xlabel('Niveau d\'intensité')
# plt.ylabel('Fréquence')

plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
# Afficher l'image pour référence
plt.figure(figsize=(8, 6))
plt.imshow(image, cmap='gray')
plt.title('Image originale')
plt.axis('off')
plt.show()

# Informations supplémentaires
print(f"Dimensions de l'image : {image.shape}")
print(f"Type de données : {image.dtype}")
print(f"Valeurs min-max : {image.min()} -
{image.max()}")
```