TP : PYTHON

→Objectifs :

- Se familiariser avec les types de données de base en Python
- Manipuler les structures de contrôle (conditions et boucles)
- Apprendre à écrire des programmes structurés avec des étapes logiques

Partie 1 : Structures de Base

1. Variables et Types de Données :

- Créer un programme qui :
- Demande le nom, l'âge, et le salaire d'un utilisateur.
- Affiche ces informations sous la forme suivante :

→ Exemple d'affichage

```
print("Nom : Alice, Âge : 25, Salaire : 3500")
```

→ Exemple de code attendu:

```
nom = input("Entrez votre nom : ")
age = int(input("Entrez votre âge : "))
salaire = float(input("Entrez votre salaire : "))
print(f"Nom : {nom}, Âge : {age}, Salaire : {salaire}")
```

2. Opérations de Base et Conversion :

- Écrire un programme qui :
- Demande à l'utilisateur deux nombres.
- Effectue les opérations suivantes : addition, soustraction, multiplication et division.
- Affiche le résultat de chaque opération.

→Exemple d'affichage attendu :

→Résultat attendu

```
print("Addition : 15, Soustraction : 5, Multiplication : 50, Division : 2.0")
```

Partie 2 : Structures de Contrôle

1. Conditions Simples :

- Écrire un programme qui demande à l'utilisateur son âge et affiche un message basé sur son âge :
- Si l'âge est inférieur à 18, affichez "Vous êtes mineur".
- Si l'âge est entre 18 et 65, affichez "Vous êtes adulte".
- Si l'âge est supérieur ou égal à 65, affichez "Vous êtes senior".

→ Exemple de code attendu :

```
age = int(input("Entrez votre âge : "))
if age < 18:
    print("Vous êtes mineur")
```

```
elif age < 65:
    print("Vous êtes adulte")
else:
    print("Vous êtes senior")
```

2. Conditions Complexes :

- Créer un programme qui demande un mot de passe et vérifie s'il est correct :
- Si le mot de passe est "python123", affichez "Accès autorisé".
- Sinon, affichez "Accès refusé".

Partie 3 : Boucles

1. Boucle `for` avec une liste :

- Écrivez un programme qui crée une liste de 5 fruits et utilise une boucle `for` pour afficher chaque fruit.

2. Boucle `while` pour deviner un nombre :

- Créer un programme qui génère un nombre aléatoire entre 1 et 10 (utiliser `import random` et `random.randint()`).
- L'utilisateur doit deviner ce nombre.
- Tant que l'utilisateur n'a pas deviné, le programme doit afficher "Essayez encore".
- Si l'utilisateur devine correctement, affichez "Félicitations, vous avez deviné !"

→ Exemple de code attendu :

```
import random
nombre_a_deviner = random.randint(1, 10)
devine = None
while devine != nombre_a_deviner:
    devine = int(input("Devinez le nombre (entre 1 et 10) : "))
    if devine == nombre_a_deviner:
        print("Félicitations, vous avez deviné !")
    else:
        print("Essayez encore")
```

3. Boucle `for` avec somme des nombres pairs :

- Écrire un programme qui :
- Utilise une boucle `for` pour parcourir les nombres de 1 à 20.
- Calcule et affiche la somme de tous les nombres pairs dans cet intervalle.

→ Exemple de code attendu :

```
somme = 0
for i in range(1, 21):
    if i % 2 == 0:
        somme += i
```

```
print("La somme des nombres pairs de 1 à 20 est :", somme)
```

Partie 4 : Exercice Final – Analyse de Notes

1. Création et saisie de notes

- Demandez à l'utilisateur combien de notes il souhaite entrer.
- Utilisez une boucle pour permettre à l'utilisateur d'entrer chaque note (entre 0 et 20).
- Stockez ces notes dans une liste.

2. Analyse des notes :

- Affichez le nombre total de notes saisies.
- Affichez la moyenne des notes.
- Affichez le nombre de notes supérieures ou égales à la moyenne calculée.

→ Exemple de code attendu :

```
notes = []
nb_notes = int(input("Combien de notes souhaitez-vous entrer ? "))
for _ in range(nb_notes):
    note = float(input("Entrez une note entre 0 et 20 : "))
    notes.append(note)
moyenne = sum(notes) / len(notes)
nb_above_moyenne = sum(1 for note in notes if note >= moyenne)
print("Nombre de notes :", len(notes))
print("Moyenne des notes :", moyenne)
print("Nombre de notes supérieures ou égales à la moyenne :", nb_above_moyenne)
```

→ Les étudiants doivent :

- Utiliser correctement les structures de base et de contrôle.
- Écrire des commentaires pour expliquer leur code.
- Gérer les erreurs éventuelles d'entrée utilisateur avec des conditions.
- Tester et s'assurer que leur programme fonctionne comme attendu.

TP1 : Manipulation des fichiers CSV avec Python et Pandas

--→ **Pré-requis** : variables, boucles, conditions, fonctions, Pandas, dictionnaires ←--

Objectifs pédagogiques

- Lire un fichier CSV en Python avec Pandas
- Analyser et manipuler des données (filtrage, tri, statistiques)
- Sauvegarder les résultats dans un nouveau fichier

- Comprendre l'importance des structures de données comme les DataFrames

Fichier de travail

Soit un fichier **CSV** nommé **etudiants.csv** contenant :

```
nom,note,age
Amine,12,21
Sara,9,20
Karim,15,22
Yasmine,8,19
Rachid,11,20
```

Partie 1 : Lecture et affichage du fichier CSV

```
import pandas as pd  # Import de la bibliothèque Pandas

df = pd.read_csv('etudiants.csv')  # Lecture du fichier CSV
print("Contenu du fichier :\n", df)  # Affichage du DataFrame
```

- import pandas as pd : importe la bibliothèque Pandas en abrégé pd.
- pd.read_csv() : lit le fichier CSV et le transforme en un DataFrame.
- print(df) : affiche tout le contenu.

Partie 2 : Traitement des données

```
# Étudiants ayant une note >= 10
admis = df[df['note'] >= 10]
print("\nÉtudiants admis :\n", admis)

# Calcul de la moyenne
moyenne = df['note'].mean()
print("\nMoyenne des notes :", moyenne)

# Trier par note décroissante
tri = df.sort_values(by='note', ascending=False)
print("\nTri décroissant :\n", tri)
```

- **df[df['note'] >= 10]** : filtre les lignes où la note est >= 10.
- **df['note'].mean()** : moyenne des valeurs de la colonne note.
- **sort_values(by='note')** : tri croissant ou décroissant.

Partie 3 : Sauvegarde

```
admis.to_csv("etudiants_admis.csv", index=False)
```

- **to_csv()** : exporte un DataFrame en CSV.
- **index=False** évite d'écrire l'index dans le fichier.

Activités à réaliser

1. Ajouter une colonne "mention" selon la note :
    - [10-12[: "Passable"
    - [12-14[: "Assez bien"
    - [14-16[: "Bien"
    - [16 et +] : "Très bien"
2. Afficher les étudiants avec mention "Bien" ou plus.
3. Afficher le nombre d'étudiants par mention.
4. Sauvegarder les résultats dans un fichier etudiants_mentions.csv.

Questions de réflexion

1. Quelle est la différence entre un dictionnaire Python et un DataFrame Pandas ?
2. Pourquoi utiliser Pandas au lieu de lire le CSV avec **open()** ?
3. Comment éviter les erreurs si le fichier CSV est mal encodé ou mal formé ?
4. Est-il possible de trier sur plusieurs colonnes ? Si oui, comment ?

Corrigé des activités

```
# Ajout de la colonne mention
def calcul_mention(note):
    if note < 10:
        return "Échec"
    elif note < 12:
        return "Passable"
    elif note < 14:
        return "Assez bien"
    elif note < 16:
        return "Bien"
    else:
        return "Très bien"

df["mention"] = df["note"].apply(calcul_mention)
print("\nAvec mentions :\n", df)

# Étudiants avec mention Bien ou Très bien
bons = df[df["mention"].isin(["Bien", "Très bien"])]
print("\nÉtudiants avec bonne mention :\n", bons)

# Nombre d'étudiants par mention
compte = df["mention"].value_counts()
print("\nNombre par mention :\n", compte)

# Sauvegarde
df.to_csv("etudiants_mentions.csv", index=False)
```

TP2 : Lecture et traitement de fichiers PDF en Python

--→ Pré-requis : boucles, conditions, fonctions, chaînes de caractères, fichiers texte ←--

Objectifs pédagogiques

- Ouvrir un fichier PDF avec Python
- Extraire le contenu texte d'un document PDF
- Analyser les données textuelles (statistiques, recherches de mots-clés)
- Sauvegarder le texte dans un fichier .txt

Préparation

Installer la bibliothèque :

pip install PyMuPDF

Fichier de travail

Télécharger un fichier PDF d'exemple, par exemple **cours_python.pdf**.

Partie 1 : Lecture et extraction de texte

```
import fitz  # alias de PyMuPDF

# Ouvrir le fichier PDF
doc = fitz.open("cours_python.pdf")

# Extraire le texte de toutes les pages
texte_total = ""
for page in doc:
    texte_total += page.get_text()

print("Extrait du texte (500 premiers caractères) :")
print(texte_total[:500])  # Affiche les 500 premiers caractères

doc.close()
```

- import fitz : importe la bibliothèque PyMuPDF.
- fitz.open() : ouvre le document PDF.
- get_text() : extrait le texte brut de la page.
- += : concatène le texte page par page.

Partie 2 : Traitement du texte

```
# Compter le nombre de mots
mots = texte_total.split()
print("Nombre total de mots :", len(mots))

# Chercher un mot-clé
mot_cle = "Python"
occurrences = texte_total.lower().count(mot_cle.lower())
```

```
print(f"Occurrences du mot '{mot_cle}' :", occurrences)
```

- split() : découpe le texte en mots selon les espaces.
- count() : compte les occurrences d'un mot (en ignorant la casse).

Partie 3 : Sauvegarde dans un fichier texte

```
# Sauvegarder le texte extrait dans un fichier .txt
with open("texte_extrait.txt", "w", encoding="utf-8") as fichier:
    fichier.write(texte_total)
print("Texte sauvegardé dans texte_extrait.txt")
```

- **with open(...)** : ouvre un fichier en mode écriture.
- **write()** : écrit le contenu dans le fichier texte.

Activités à réaliser

1. Écrire une fonction qui compte les 10 mots les plus fréquents dans le texte (hors mots vides comme *le*, *la*, *de*, *et*, etc.).
2. Créer un dictionnaire {mot: nb_occurrences} trié par fréquence décroissante.
3. Afficher tous les mots apparaissant plus de 10 fois.
4. Enregistrer ce résultat dans un fichier **statistiques.txt**.

Questions de réflexion

1. Pourquoi **get_text()** ne fonctionne-t-il pas toujours sur certains PDF ?
2. Quelle est la différence entre un OCR et une extraction classique ?
3. Comment peut-on détecter si un PDF contient du texte ou des images uniquement ?
4. Quelle serait l'utilité de cette technique dans un contexte professionnel ?

Corrigé des activités

```
from collections import Counter
import re

# Nettoyer le texte
texte_min = texte_total.lower()
texte_sans_pontuation = re.sub(r"[^\w\s]", "", texte_min)  # Retire ponctuations
mots = texte_sans_pontuation.split()

# Liste de mots à ignorer (mots vides)
stopwords = ["le", "la", "les", "de", "des", "et", "en", "à", "un", "une", "du", "dans", "est", "pour", "avec", "par"]

# Filtrer
mots_filtres = [mot for mot in mots if mot not in stopwords]

# Compter les mots
frequence = Counter(mots_filtres)

# Afficher les 10 plus fréquents
print("10 mots les plus fréquents :")
for mot, count in frequence.most_common(10):
```

```
    print(f"{mot} : {count}")

# Mots avec plus de 10 occurrences
mots_plus10 = {mot: count for mot, count in frequence.items() if count > 10}

# Sauvegarde dans un fichier
with open("statistiques.txt", "w", encoding="utf-8") as f:
    for mot, count in mots_plus10.items():
        f.write(f"{mot} : {count}\n")
```

TP3 : Lecture et traitement d'images avec Python

--→ Pré-requis : Python, fonctions, boucles, NumPy, notion de matrices et tableaux ←--

Objectifs pédagogiques

- Charger une image avec OpenCV
- Afficher une image dans une fenêtre
- Appliquer des traitements de base : redimensionnement, passage en niveaux de gris, flou, détection de contours
- Sauvegarder une image modifiée

Installation des bibliothèques nécessaires

pip install opencv-python

Fichier de travail

Utilisez une image locale, par exemple : **chat.jpg** (ou toute autre image dans votre dossier de travail).

Partie 1 : Lecture et affichage d'une image

```
import cv2  # OpenCV

# Charger l'image
img = cv2.imread("chat.jpg")

# Afficher les dimensions
print("Dimensions de l'image :", img.shape)

# Affichage de l'image
cv2.imshow("Image Originale", img)
cv2.waitKey(0)   # Attend une touche
cv2.destroyAllWindows()
```

- **cv2.imread()** : lit l'image.
- **img.shape** : donne les dimensions (hauteur, largeur, canaux).

- **cv2.imshow()** : affiche l'image dans une fenêtre.
- **cv2.waitKey(0)** : attend que l'utilisateur appuie sur une touche.
- **cv2.destroyAllWindows()** : ferme toutes les fenêtres ouvertes.

Partie 2 : Traitement de l'image

```
# Redimensionner l'image
img_resize = cv2.resize(img, (300, 300))

# Convertir en niveaux de gris
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Appliquer un flou gaussien
flou = cv2.GaussianBlur(gray, (7, 7), 0)

# Détection des contours avec Canny
bords = cv2.Canny(flou, 50, 150)

# Affichage de toutes les versions
cv2.imshow("Redimensionnée", img_resize)
cv2.imshow("Gris", gray)
cv2.imshow("Flou", flou)
cv2.imshow("Contours", bords)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **resize()** : redimensionne l'image.
- **cvtColor()** : convertit l'image en niveaux de gris.
- **GaussianBlur()** : ajoute un effet de flou.
- **Canny()** : détecte les bords.

Partie 3 : Sauvegarde des images traitées

```
cv2.imwrite("chat_gris.jpg", gray)
cv2.imwrite("chat_bords.jpg", bords)
print("Images sauvegardées.")
```

Activités à réaliser

1. Charger une autre image (chien.jpg, voiture.jpg, etc.)
2. Appliquer les étapes suivantes :
    - Conversion en gris
    - Flou
    - Détection des bords
    - Affichage de tous les résultats
3. Afficher les dimensions de chaque image à chaque étape.
4. Sauvegarder les résultats sous les noms :
    - **nom_image_gris.jpg**
    - **nom_image_bords.jpg**

Questions de réflexion

1. Quelle est la différence entre une image couleur et une image en niveaux de gris (au niveau des matrices) ?
2. Pourquoi appliquer un flou avant la détection de contours ?
3. Que se passe-t-il si on change les paramètres de la fonction Canny ?
4. Quelle est la structure d'une image en mémoire en Python avec OpenCV ?

Corrigé des activités

```
img = cv2.imread("chien.jpg")
print("Original :", img.shape)

# Gris
gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
print("Gris :", gris.shape)

# Flou
flou = cv2.GaussianBlur(gris, (5, 5), 0)
print("Flou :", flou.shape)

# Contours
bords = cv2.Canny(flou, 100, 200)
print("Contours :", bords.shape)

# Affichage
cv2.imshow("Gris", gris)
cv2.imshow("Bords", bords)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Sauvegarde
cv2.imwrite("chien_gris.jpg", gris)
cv2.imwrite("chien_bords.jpg", bords)
```

TP4 : Lecture et traitement de vidéos avec Python (OpenCV)

--→ Pré-requis : Images, boucles, fonctions, OpenCV ←--

Objectifs pédagogiques

- Lire une vidéo image par image avec OpenCV
- Afficher une vidéo image par image
- Appliquer des traitements : niveaux de gris, détection de mouvement simple
- Enregistrer une nouvelle vidéo traitée

Installation des bibliothèques nécessaires

pip install opencv-python

Fichier de travail

Utilisez une vidéo locale dans votre dossier, par exemple : video.mp4.

Partie 1 : Lecture et affichage image par image

```
import cv2

# Chargement de la vidéo
cap = cv2.VideoCapture("video.mp4")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    cv2.imshow("Vidéo originale", frame)

    # Quitter si la touche 'q' est appuyée
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

- VideoCapture() : ouvre la vidéo.
- read() : lit chaque frame (image).
- imshow() : affiche la frame.
- waitKey(25) : attend 25ms, imitant le taux de 40 FPS.

Partie 2 : Traitement de la vidéo (conversion en gris)

```
cap = cv2.VideoCapture("video.mp4")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Conversion en niveaux de gris
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow("Vidéo en niveaux de gris", gray)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Partie 3 : Sauvegarde d'une vidéo traitée

```
cap = cv2.VideoCapture("video.mp4")
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter("output.avi", fourcc, 20.0, (640, 480), isColor=False)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(gray, (640, 480))

    out.write(resized)
    cv2.imshow("Traitée", resized)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Partie 4 : Détection de mouvement simple

```
cap = cv2.VideoCapture("video.mp4")
_, frame1 = cap.read()
_, frame2 = cap.read()

while cap.isOpened():
    diff = cv2.absdiff(frame1, frame2)
    gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5,5), 0)
    _, thresh = cv2.threshold(blur, 20, 255, cv2.THRESH_BINARY)
    dilated = cv2.dilate(thresh, None, iterations=2)

    cv2.imshow("Mouvement détecté", dilated)

    frame1 = frame2
    ret, frame2 = cap.read()
    if not ret or cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Activités à réaliser

1. Charger une autre vidéo (cam.mp4, voiture.mp4, etc.)
2. Convertir chaque frame en niveaux de gris
3. Appliquer un flou puis une détection de bords (Canny)
4. Afficher chaque image traitée en temps réel
5. Enregistrer le résultat dans une nouvelle vidéo (video_traitée.avi)

Questions de réflexion

1. Quelle est la différence entre une vidéo et une série d’images ?
2. Pourquoi utiliser `cv2.waitKey(25)` dans la boucle ?
3. Quel est le rôle de `cv2.VideoWriter_fourcc` ?
4. Pourquoi appliquer un flou avant de détecter les mouvements ou les contours ?
5. Quelles seraient les limites de la détection de mouvement avec cette méthode ?

Corrigé des activités

```
cap = cv2.VideoCapture("cam.mp4")
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter("video_traitee.avi", fourcc, 20.0, (640, 480), isColor=False)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    flou = cv2.GaussianBlur(gray, (5, 5), 0)
    contours = cv2.Canny(flou, 50, 150)

    resized = cv2.resize(contours, (640, 480))
    out.write(resized)

    cv2.imshow("Vidéo traitée", resized)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```