R 5.A.06

2025 – 2026

Sensibilisation à la programmation multimédia

Corrigé du TP n° 3
Vidéo

ANNE Jean-François

Sensibilisation à la programmation multimédia

Le but de ce TP est de se familiariser avec le Sensibilisation à la programmation multimédia.

A. Exercices de base en vidéo :

Il faut utiliser la version 1.0.3 de moviepy, et si cela ne fonctionnes pas utiliser pillow en version 9.5.0.

1°) Exercice 1 :

Ecrire un programme python qui lit et affiche les caractéristiques d'une vidéo.

```
from moviepy.editor import VideoFileClip

def get_video_characteristics(video_path):
    try:
        # Ouvrir le fichier vidéo en utilisant VideoFileClip
        video = VideoFileClip(video_path)

        resolution = f"{video.size[0]}x{video.size[1]} pixels"
        duration = video.duration
        fps = video.fps
        nframes = video.reader.nframes

        print("Caractéristiques de la vidéo :")
        print(f"Résolution: {resolution}")
        print(f"Durée: {duration:.2f} secondes")
        print(f"FPS: {fps:.2f}")
        print(f"Nombre de trames: { nframes }")
    except Exception as e:
        print("Une erreur s'est produite :", e)

# Chemin du fichier vidéo
video_path = "music.mp4"

# Appeler la fonction pour obtenir les caractéristiques de la vidéo
get_video_characteristics(video_path)
```

2°) Exercice 2 :

Ecrire un programme python qui extrait l'audio en mp3 d'un fichier vidéo.mp4 fourni.

```
from moviepy.editor import *

def extract_audio_from_video(input_video_path, output_audio_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Extraire l'audio
        audio = video.audio

        # Sauvegarder l'audio en MP3
        audio.write_audiofile(output_audio_path)
```

```
            # Fermer le fichier vidéo
            video.close()

            print("Extraction audio terminée.")
        except Exception as e:
            print("Une erreur s'est produite :", e)

    # Chemin du fichier vidéo d'entrée
    input_video_path = "music.mp4"

    # Chemin de sauvegarde du fichier audio MP3
    output_audio_path = "audio.mp3"

    # Appeler la fonction pour extraire l'audio
    extract_audio_from_video(input_video_path, output_audio_path)
```

3°) ***Exercice 3 :***

Ecrire un programme python qui double la résolution en X et en Y d'un fichier vidéo.mp4 fourni.

```
from moviepy.editor import *
from PIL import Image

# patch for broken PIL version
from pkg_resources import parse_version
Image.ANTIALIAS=Image.LANCZOS

def double_resolution(input_video_path, output_video_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Doubler la résolution en X et en Y
        new_width = video.w * 2
        new_height = video.h * 2

        # Créer un nouveau clip avec la résolution doublée
        new_video = video.resize((new_width, new_height))

        # Sauvegarder le nouveau clip vidéo
        new_video.write_videofile(output_video_path, codec='libx264')

        # Fermer les fichiers vidéo
        video.close()
        new_video.close()

        print("Doubler la résolution terminée.")
    except Exception as e:
        print("Une erreur s'est produite :", e)

# Chemin du fichier vidéo d'entrée
```

```
input_video_path = "music.mp4"

# Chemin de sauvegarde du nouveau fichier vidéo
output_video_path = "2music.mp4"

# Appeler la fonction pour doubler la résolution
double_resolution(input_video_path, output_video_path)
```

4°) ***Exercice 4 :***

Ecrire un programme python qui transforme la vidéo fournie en niveau de gris.

```
from moviepy.editor import *

def video_to_grayscale(input_video_path, output_video_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Convertir la vidéo en niveau de gris
        grayscale_video = video.fx(vfx.colorx, 0)

        # Sauvegarder la vidéo en niveau de gris
        grayscale_video.write_videofile(output_video_path, codec='libx264')

        # Fermer les fichiers vidéo
        video.close()
        grayscale_video.close()

        print("Conversion en niveau de gris terminée.")
    except Exception as e:
        print("Une erreur s'est produite :", e)

# Chemin du fichier vidéo d'entrée
input_video_path = "music.mp4"

# Chemin de sauvegarde du nouveau fichier vidéo en niveau de gris
output_video_path = "music_gris.mp4"

# Appeler la fonction pour convertir la vidéo en niveau de gris
video_to_grayscale(input_video_path, output_video_path)
```

5°) ***Exercice 5 :***

Ecrire un programme python qui transforme la vidéo fournie de 16/9 en 4/3.

```
from moviepy.editor import *

def video_16_9_to_4_3(input_video_path, output_video_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Définir les dimensions pour le format 4/3
        width_4_3 = int(video.h * 4 / 3)
```

```
            height_4_3 = video.h

            # Recadrer la vidéo au format 4/3
            cropped_video = video.crop(x_center=(video.w - width_4_3) / 2,
                                       y_center=video.h / 2,
                                       width=width_4_3,
                                       height=height_4_3)

            # Sauvegarder la vidéo recadrée
            cropped_video.write_videofile(output_video_path, codec='libx264')

            # Fermer les fichiers vidéo
            video.close()
            cropped_video.close()

            print("Conversion au format 4/3 terminée.")
        except Exception as e:
            print("Une erreur s'est produite :", e)

    # Chemin du fichier vidéo d'entrée
    input_video_path = "music.mp4"

    # Chemin de sauvegarde du nouveau fichier vidéo au format 4/3
    output_video_path = "music_4_3.mp4"

    # Appeler la fonction pour convertir la vidéo en format 4/3
    video_16_9_to_4_3(input_video_path, output_video_path)
```

6°) ***Exercice 6 :***

Ecrire un programme python qui transforme la vidéo fournie de 4/3 en 16/9.

```
from moviepy.editor import *

def video_4_3_to_16_9(input_video_path, output_video_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Définir les dimensions pour le format 16/9
        width_16_9 = int(video.h * 16 / 9)
        height_16_9 = video.h

        # Recadrer la vidéo au format 16/9
        cropped_video = video.crop(x_center=(video.w - width_16_9) / 2,
                                   y_center=video.h / 2,
                                   width=width_16_9,
                                   height=height_16_9)

        # Ajouter des bandes noires en haut et en bas pour garder le format 16/9
        final_video = cropped_video.margin(top=(video.h - height_16_9) // 2,
                                           bottom=(video.h - height_16_9) // 2,
                                           left=0, right=0, color=(0, 0, 0))

        # Sauvegarder la vidéo recadrée
        final_video.write_videofile(output_video_path, codec='libx264')

        # Fermer les fichiers vidéo
        video.close()
        cropped_video.close()
        final_video.close()
```

```
            print("Conversion au format 16/9 terminée.")
        except Exception as e:
            print("Une erreur s'est produite :", e)

    # Chemin du fichier vidéo d'entrée
    input_video_path = "music.mp4"

    # Chemin de sauvegarde du nouveau fichier vidéo au format 16/9
    output_video_path = "music_16_9.mp4"

    # Appeler la fonction pour convertir la vidéo en format 16/9
    video_4_3_to_16_9(input_video_path, output_video_path)
```

7°) ***Exercice 7 :***

Ecrire un programme python qui convertit une vidéo fournie au format DVD en format Blu-ray.

```
import moviepy.editor as mp

def video_dvd_to_bluray(input_video_path, output_video_path):
    try:
        # Charger le fichier vidéo DVD
        dvd_video = mp.VideoFileClip(input_video_path)

        # Définir les dimensions pour le format Blu-ray (1080p)
        width_bluray = 1920
        height_bluray = 1080

        # Redimensionner la vidéo au format Blu-ray
        bluray_video = dvd_video.resize((width_bluray, height_bluray))

        # Encoder la vidéo en utilisant le codec H.264 pour le format Blu-ray
        bluray_video.write_videofile(
            output_video_path, codec='libx264', fps=24)

        # Fermer les fichiers vidéo
        dvd_video.close()
        bluray_video.close()

        print("Conversion au format Blu-ray terminée.")
    except Exception as e:
        print("Une erreur s'est produite :", e)

# Chemin du fichier vidéo au format DVD
input_video_path = "music.mp4"

# Chemin de sauvegarde du nouveau fichier vidéo au format Blu-ray
output_video_path = "music_bluray.mp4"

# Appeler la fonction pour convertir la vidéo au format Blu-ray
video_dvd_to_bluray(input_video_path, output_video_path)
```

8°) ***Exercice 8 :***

Ecrire un programme python qui convertit une vidéo fournie en couleur sépia imitant un vieux film.

```
from moviepy.editor import *
import numpy as np

def apply_sepia_filter(image):
    # Convertir l'image en tableau de pixels
```

```
    pixels = np.array(image)

    # Convertir l'image en niveaux de gris
    gray_pixels = np.dot(pixels[..., :3], [0.299, 0.587, 0.114])

    # Appliquer l'effet sépia en ajustant les canaux de couleur RVB
    sepia_pixels = np.zeros_like(pixels)
    sepia_pixels[:, :, 0] = np.clip(
        gray_pixels * 0.393 + gray_pixels * 0.769 + gray_pixels * 0.189, 0, 255)
    sepia_pixels[:, :, 1] = np.clip(
        gray_pixels * 0.349 + gray_pixels * 0.686 + gray_pixels * 0.168, 0, 255)
    sepia_pixels[:, :, 2] = np.clip(
        gray_pixels * 0.272 + gray_pixels * 0.534 + gray_pixels * 0.131, 0, 255)

    return sepia_pixels.astype(np.uint8)

def video_to_sepiatone(input_video_path, output_video_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Appliquer l'effet sépia à chaque image de la vidéo
        sepia_video = video.fl_image(apply_sepia_filter)

        # Sauvegarder la vidéo en couleur marron
        sepia_video.write_videofile(output_video_path, codec='libx264')

        # Fermer les fichiers vidéo
        video.close()
        sepia_video.close()

        print("Transformation en couleur marron terminée.")
    except Exception as e:
        print("Une erreur s'est produite :", e)

# Chemin du fichier vidéo d'entrée
input_video_path = "Les Temps Modernes.mp4"

# Chemin de sauvegarde du nouveau fichier vidéo en couleur marron
output_video_path = "Les Temps Modernes_sepia.mp4"

# Appeler la fonction pour transformer la vidéo en couleur marron
video_to_sepiatone(input_video_path, output_video_path)
```

9°) ***Exercice 9 :***

Ecrire un programme python qui insère le fichier de sous titres dans la vidéo fournie.

```
from moviepy.editor import *
from moviepy.tools import subprocess_call
from moviepy.video.tools.subtitles import TextClip, SubtitlesClip
import moviepy.config as cfg

# Spécifiez le chemin vers l'exécutable d'ImageMagick ici
```

```
# Pour les utilisateurs de Windows
subprocess_call(
    ["C:\Program Files (x86)\ImageMagick-7.1.1-Q16\convert.exe", "-version"])

# Pour les utilisateurs de Mac ou Linux
# subprocess_call(["/chemin/vers/votre/convert", "-version"])

def add_subtitles(input_video_path, output_video_path, subtitles_path):
    try:
        # Charger le fichier vidéo
        video = VideoFileClip(input_video_path)

        # Charger le fichier de sous-titres au format SRT (SubRip)
        def generator(txt): return TextClip(
            txt, font='Arial',  fontsize=24, color='white')

        subtitles = SubtitlesClip(subtitles_path, generator)
        # subtitles = SubtitlesClip(subtitles_path, fontsize=24, color='white')

        # Positionner les sous-titres au bas de l'écran
        subtitles = subtitles.set_position(("center", "bottom"))

        # Ajouter les sous-titres à la vidéo
        video_with_subtitles = CompositeVideoClip([video, subtitles])

        # Sauvegarder la vidéo avec les sous-titres
        video_with_subtitles.write_videofile(
            output_video_path, codec='libx264')

        # Fermer les fichiers vidéo
        video.close()
        video_with_subtitles.close()

        print("Sous-titres ajoutés à la vidéo.")
    except Exception as e:
        print("Une erreur s'est produite :", e)

# Chemin du fichier vidéo d'entrée
input_video_path = "music.mp4"

# Chemin du fichier de sous-titres au format SRT (SubRip)
subtitles_path = "sous-titres.srt"

# Chemin de sauvegarde du nouveau fichier vidéo avec les sous-titres
output_video_path = "video_sous-titres.mp4"

# Appeler la fonction pour insérer et afficher les sous-titres dans la vidéo
add_subtitles(input_video_path, output_video_path, subtitles_path)
```

Problème de Imagemagick à résoudre :

I'm on Windows 10. These are the last few lines of the file config_defaults.py on my computer. You may notice that ImageMagick, including its binaries, is stored in a location away from other scripts. config_defaults.py is stored in the folder C:\Python34\Lib\site-packages\moviepy.

```
IMAGEMAGICK_BINARY
    For linux users, 'convert' should be fine.
    For Windows users, you must specify the path to the ImageMagick
    'convert' binary. For instance:

    IMAGEMAGICK_BINARY = r"C:\Program Files\ImageMagick-6.8.8-Q16\convert.exe"

"""

FFMPEG_BINARY = 'ffmpeg-imageio'
#~ IMAGEMAGICK_BINARY = 'auto-detect'
IMAGEMAGICK_BINARY = "C:\\Program Files\\ImageMagick-7.0.4-Q16\\magick.exe"
```

Redémarrer vscode au minimum

B. Création de textes en vidéo :

En utilisant le site :

- *https://educationdunumerique.fr/moviepy.php*

Testez les différents exemples de texte.

1°) Lettres qui se dispersent :

```python
# dispersion_lettres.py
import numpy as np
from moviepy.editor import *
from moviepy.video.tools.segmenting import findObjects
# Création du texte et position centrée,
ecran = (600,600)
le_texte = TextClip("Affichage\n de lettres", color='gold', font="verdana-bold",
kerning=5 , transparent=True, stroke_color='red', stroke_width=4, fontsize=90)
cvc = CompositeVideoClip([le_texte.set_pos('center')], size=ecran)
# Fonctions pour déplacer les lettres
deplace=lambda a: np.array([[np.cos(a), np.sin(a)],[-np.sin(a), np.cos(a)]])

def reunion(screenpos, i, nletters):
    d= lambda t : 1.0/(0.3+t**8)
    a = i*np.pi / nletters
    v = deplace(a).dot([-1, 0])
    if i % 2:
        v[1] = -v[1]
    return lambda t: screenpos+400*d(t)*deplace(0.5*d(t)*a).dot(v)

def cascade(screenpos, i, nletters):
    v=np.array([0, -1])
    d=lambda t: 1 if t < 0 else abs(np.sinc(t)/(1+t**4))
    return lambda t: screenpos+v*400*d(t-0.15*i)

def translation(screenpos, i, nletters):
    v = np.array([0, -1])
    d=lambda t: max(0,3-3*t)
    return lambda t: screenpos-v*400*d(t-0.2*i)

def dispersion(screenpos, i, nletters):
    d=lambda t: max(0, t)
    a = i*np.pi/nletters
```

```
    v = deplace(a).dot([-1, 0])
    if i % 2: v[1] = -v[1]
    return lambda t: screenpos+400*d(t-0.1*i)*deplace(-0.2*d(t)*a).dot(v)
lettres=findObjects(cvc) # une liste

def lettrebouge(lettres, funcpos):  # Animation des lettres
    return [lettre.set_pos(funcpos(lettre.screenpos, i, len(lettres)))
            for i, lettre in enumerate(lettres)]

liste_video=[CompositeVideoClip(lettrebouge(lettres, funcpos),
        size=ecran).subclip(0, 2).on_color(ecran, (255, 255, 255),
        col_opacity=1)
        for funcpos in [dispersion]]
    # concaténation et sauvegarde de la vidéo
film=concatenate_videoclips(liste_video)
film.write_videofile( 'dispersion.mp4' ,fps=25,codec='mpeg4' )
```

2°) ***Lettres qui se réunissent :***

```
# réunion_lettres.py
import numpy as np
from moviepy.editor import *
from moviepy.video.tools.segmenting import findObjects
# Création du texte et position centrée,
ecran = (600,600)
le_texte = TextClip("Affichage\n de lettres", color='gold', font="verdana-bold",
kerning=5 , transparent=True, stroke_color='red', stroke_width=4, fontsize=90)
cvc = CompositeVideoClip([le_texte.set_pos('center')], size=ecran)
# Fonctions pour déplacer les lettres
deplace=lambda a: np.array([[np.cos(a), np.sin(a)],[-np.sin(a), np.cos(a)]])

def reunion(screenpos, i, nletters):
    d= lambda t : 1.0/(1.5+t**8)
    a = i*np.pi / nletters
    v = deplace(a).dot([-1, 0])
    if i % 2:
        v[1] = -v[1]
    return lambda t: screenpos+400*d(t)*deplace(0.5*d(t)*a).dot(v)

def cascade(screenpos, i, nletters):
    v=np.array([0, -1])
    d=lambda t: 1 if t < 0 else abs(np.sinc(t)/(1+t**4))
    return lambda t: screenpos+v*400*d(t-0.15*i)

def translation(screenpos, i, nletters):
    v = np.array([0, -1])
    d=lambda t: max(0,3-3*t)
    return lambda t: screenpos-v*400*d(t-0.2*i)

def dispersion(screenpos, i, nletters):
    d=lambda t: max(0, t)
    a = i*np.pi/nletters
    v = deplace(a).dot([-1, 0])
```

```
    if i % 2: v[1] = -v[1]
    return lambda t: screenpos+400*d(t-0.1*i)*deplace(-0.2*d(t)*a).dot(v)
lettres=findObjects(cvc) # une liste

def lettrebouge(lettres, funcpos):  # Animation des lettres
    return [lettre.set_pos(funcpos(lettre.screenpos, i, len(lettres)))
            for i, lettre in enumerate(lettres)]

liste_video=[CompositeVideoClip(lettrebouge(lettres, funcpos),
        size=ecran).subclip(0, 4).on_color(ecran, (255, 255, 255),
        col_opacity=1)
        for funcpos in [reunion]]
    # concaténation et sauvegarde de la vidéo
film=concatenate_videoclips(liste_video)
film.write_videofile( 'reunion.mp4' ,fps=25,codec='mpeg4' )
```

3°) ***Lettres en cascade :***

```
# cascade_lettres.py
import numpy as np
from moviepy.editor import *
from moviepy.video.tools.segmenting import findObjects
# Création du texte et position centrée,
ecran = (600,600)
le_texte = TextClip("Affichage\n de lettres", color='gold', font="verdana-bold",
kerning=5 , transparent=True, stroke_color='red', stroke_width=4, fontsize=90)
cvc = CompositeVideoClip([le_texte.set_pos('center')], size=ecran)
# Fonctions pour déplacer les lettres
deplace=lambda a: np.array([[np.cos(a), np.sin(a)],[-np.sin(a), np.cos(a)]])

def reunion(screenpos, i, nletters):
    d= lambda t : 1.0/(0.3+t**8)
    a = i*np.pi / nletters
    v = deplace(a).dot([-1, 0])
    if i % 2:
        v[1] = -v[1]
    return lambda t: screenpos+400*d(t)*deplace(0.5*d(t)*a).dot(v)

def cascade(screenpos, i, nletters):
    v=np.array([0, -1])
    d=lambda t: 1 if t < 0 else abs(np.sinc(t)/(1+t**4))
    return lambda t: screenpos+v*400*d(t-0.15*i)

def translation(screenpos, i, nletters):
    v = np.array([0, -1])
    d=lambda t: max(0,3-3*t)
    return lambda t: screenpos-v*400*d(t-0.2*i)

def dispersion(screenpos, i, nletters):
    d=lambda t: max(0, t)
    a = i*np.pi/nletters
    v = deplace(a).dot([-1, 0])
    if i % 2: v[1] = -v[1]
```

```
    return lambda t: screenpos+400*d(t-0.1*i)*deplace(-0.2*d(t)*a).dot(v)
      lettres=findObjects(cvc) # une liste

def lettrebouge(lettres, funcpos):  # Animation des lettres
    return [lettre.set_pos(funcpos(lettre.screenpos, i, len(lettres)))
            for i, lettre in enumerate(lettres)]

liste_video=[CompositeVideoClip(lettrebouge(lettres, funcpos),
        size=ecran).subclip(0, 4).on_color(ecran, (255, 255, 255),
        col_opacity=1)
        for funcpos in [cascade]]
    # concaténation et sauvegarde de la vidéo
film=concatenate_videoclips(liste_video)
film.write_videofile( 'cascade.mp4' ,fps=25,codec='mpeg4' )
```

4°) ***Lettres en translations :***

```
# 4604_translation_lettres.py
import numpy as np
from moviepy.editor import *
from moviepy.video.tools.segmenting import findObjects
# Création du texte et position centrée,
ecran = (600,600)
le_texte = TextClip("Affichage\n de lettres", color='gold', font="verdana-bold",
kerning=5 , transparent=True, stroke_color='red', stroke_width=4, fontsize=90)
cvc = CompositeVideoClip([le_texte.set_pos('center')], size=ecran)
# Fonctions pour déplacer les lettres
deplace=lambda a: np.array([[np.cos(a), np.sin(a)],[-np.sin(a), np.cos(a)]])

def reunion(screenpos, i, nletters):
    d= lambda t : 1.0/(0.3+t**8)
    a = i*np.pi / nletters
    v = deplace(a).dot([-1, 0])
    if i % 2:
        v[1] = -v[1]
    return lambda t: screenpos+400*d(t)*deplace(0.5*d(t)*a).dot(v)

def cascade(screenpos, i, nletters):
    v=np.array([0, -1])
    d=lambda t: 1 if t < 0 else abs(np.sinc(t)/(1+t**4))
    return lambda t: screenpos+v*400*d(t-0.15*i)

def translation(screenpos, i, nletters):
    v = np.array([0, -1])
    d=lambda t: max(0,3-3*t)
    return lambda t: screenpos-v*400*d(t-0.2*i)

def dispersion(screenpos, i, nletters):
    d=lambda t: max(0, t)
    a = i*np.pi/nletters
    v = deplace(a).dot([-1, 0])
    if i % 2: v[1] = -v[1]
    return lambda t: screenpos+400*d(t-0.1*i)*deplace(-0.2*d(t)*a).dot(v)
```

```
lettres=findObjects(cvc) # une liste

def lettrebouge(lettres, funcpos):  # Animation des lettres
    return [lettre.set_pos(funcpos(lettre.screenpos, i, len(lettres)))
            for i, lettre in enumerate(lettres)]

liste_video=[CompositeVideoClip(lettrebouge(lettres, funcpos),
        size=ecran).subclip(0, 6).on_color(ecran, (255, 255, 255),
        col_opacity=1)
        for funcpos in [translation]]
    # concaténation et sauvegarde de la vidéo
film=concatenate_videoclips(liste_video)
film.write_videofile( 'translation.mp4' ,fps=25,codec='mpeg4' )
```

5°) ***Tous :***

```
# 4604_tous_lettres.py
import numpy as np
from moviepy.editor import *
from moviepy.video.tools.segmenting import findObjects
# Création du texte et position centrée,
ecran = (600,600)
le_texte = TextClip("Affichage\n de lettres", color='gold', font="verdana-bold",
kerning=5 , transparent=True, stroke_color='red', stroke_width=4, fontsize=90)
cvc = CompositeVideoClip([le_texte.set_pos('center')], size=ecran)
# Fonctions pour déplacer les lettres
deplace=lambda a: np.array([[np.cos(a), np.sin(a)],[-np.sin(a), np.cos(a)]])

def reunion(screenpos, i, nletters):
    d= lambda t : 1.0/(0.3+t**8)
    a = i*np.pi / nletters
    v = deplace(a).dot([-1, 0])
    if i % 2:
        v[1] = -v[1]
    return lambda t: screenpos+400*d(t)*deplace(0.5*d(t)*a).dot(v)

def cascade(screenpos, i, nletters):
    v=np.array([0, -1])
    d=lambda t: 1 if t < 0 else abs(np.sinc(t)/(1+t**4))
    return lambda t: screenpos+v*400*d(t-0.15*i)

def translation(screenpos, i, nletters):
    v = np.array([0, -1])
    d=lambda t: max(0,3-3*t)
    return lambda t: screenpos-v*400*d(t-0.2*i)

def dispersion(screenpos, i, nletters):
    d=lambda t: max(0, t)
    a = i*np.pi/nletters
    v = deplace(a).dot([-1, 0])
    if i % 2: v[1] = -v[1]
    return lambda t: screenpos+400*d(t-0.1*i)*deplace(-0.2*d(t)*a).dot(v)
lettres=findObjects(cvc) # une liste
```

```
def lettrebouge(lettres, funcpos):  # Animation des lettres
    return [lettre.set_pos(funcpos(lettre.screenpos, i, len(lettres)))
            for i, lettre in enumerate(lettres)]

liste_video=[CompositeVideoClip(lettrebouge(lettres, funcpos),
        size=ecran).subclip(0, 6).on_color(ecran, (255, 255, 255),
        col_opacity=1)
        for funcpos in [dispersion, translation, cascade, reunion]]
    # concaténation et sauvegarde de la vidéo
film=concatenate_videoclips(liste_video)
film.write_videofile( 'tout.mp4' ,fps=25,codec='mpeg4' )
```

C. ***Synthèse :***

Ecrire un programme python qui prendre un fichier vidéo, lui met au début une page de présentation de la vidéo, insérer un fichier de sous-titres au format SRT dans la vidéo, puis faire un générique de fin. Sauvegarder la vidéo ainsi générée dans un fichier en mp4.

```
from moviepy.editor import *


def create_movie_credits(artists, director, duration, output_video_path):
    # Création du texte des génériques
    credits_text = f"Artistes : {artists}\nRéalisateur : {director}\nDurée du film :
{duration} minutes"

    # Taille de l'écran et police du texte
    ecran = (800, 600)
    font_size = 40

    # Création de la fonction pour animer le texte défilant
    def scroll_text(t):
        h, w = ecran[1], ecran[0]
        text_h = font_size
        y_position = max(0, int((h - text_h) * t / text_duration))
        return ('center', y_position)

    # Durée du texte défilant
    text_duration = 10

    # Création de la composition personnalisée
    final_video = CompositeVideoClip([
        ColorClip(ecran, color=(0, 0, 0), duration=text_duration),
        TextClip(credits_text, color='white', font="Arial",
                 fontsize=font_size, method='label', interline=30)
        .set_position(scroll_text)
        .set_duration(text_duration)
        .crossfadein(1)
        .crossfadeout(1),
        ColorClip(ecran, color=(0, 0, 0), duration=2)
    ])

    # Sauvegarde de la vidéo
    final_video.write_videofile(
```

```
        output_video_path, codec='libx264', fps=25, threads=4)

    print("Générique de film généré avec succès.")


# Informations pour le générique
artists_list = "Acteur 1, Acteur 2, Actrice 1"
director_name = "Réalisateur X"
film_duration = 120  # Durée du film en minutes

# Chemin de sauvegarde du générique de film
output_video_path = "generique_film.mp4"

# Appeler la fonction pour générer le générique de film
create_movie_credits(artists_list, director_name,
                     film_duration, output_video_path)
```

D. Webographie :

- https://www.geeksforgeeks.org/moviepy-applying-color-effect-on-video-clip/
- https://downsub.com/?url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DdRPR5OseAjc
- https://www.youtube.com/watch?v=ISiCDsu405A
- https://github.com/Zulko/moviepy/issues/378
- https://www.youtube.com/watch?v=h4rdulAGbbQ