MoviePy v2.0 have introduced breaking changes, see Updating from v1.X to v2.X for more info.

Loading resources as clips

The first step for making a video with MoviePy is to load the resources you wish to include in the final video.

In this section we present the different sorts of clips and how to load them. For information on modifying a clip, see Modifying clips and apply effects. For how to put clips together see Compositing multiple clips. And for how to see/save theme, see Previewing and saving video clips (we will usually save them in example, but we wont explain here).

There's a lot of different resources you can use with MoviePy, and you will load different resources with different subtypes of `Clip`, and more precisely of `AudioClip` for any audio element, or `VideoClip` for any visual element.

The following code summarizes the base clips that you can create with moviepy:

```
from moviepy import (
    VideoClip,
    VideoFileClip,
    ImageSequenceClip,
    ImageClip,
    TextClip,
    ColorClip,
    AudioFileClip,
    AudioClip,
)
import numpy as np

# Define some constants for later use
black = (255, 255, 255)  # RGB for black


def frame_function(t):
    """Random noise image of 200x100"""
    return np.random.randint(low=0, high=255, size=(100, 200, 3))
```

Skip to main content

```
def frame_function_audio(t):
    """A note by producing a sinewave of 440 Hz"""
    return np.sin(440 * 2 * np.pi * t)


# Now lets see how to load different type of resources !

# VIDEO CLIPS
# for custom animations, where frame_function is a function returning an image
# as numpy array for a given time
clip = VideoClip(frame_function, duration=5)
clip = VideoFileClip("example.mp4")  # for videos
# for a list or directory of images to be used as a video sequence
clip = ImageSequenceClip("example_img_dir", fps=24)
clip = ImageClip("example.png")  # For a picture
# To create the image of a text
clip = TextClip(font="./example.ttf", text="Hello!", font_size=70, color="black")
# a clip of a single unified color, where color is a RGB tuple/array/list
clip = ColorClip(size=(460, 380), color=black)

# AUDIO CLIPS
# for audio files, but also videos where you only want the keep the audio track
clip = AudioFileClip("example.wav")
# for custom audio, where frame_function is a function returning a
# float (or tuple for stereo) for a given time
clip = AudioClip(frame_function_audio, duration=3)
```

The best to understand all these clips more thoroughly is to read the full documentation for each in the Api Reference.

Releasing resources by closing a clip

When you create some types of clip instances - e.g. `VideoFileClip` or `AudioFileClip` - MoviePy creates a subprocess and locks the file. In order to release these resources when you are finished you should call the `close()` method.

This is more important for more complex applications and is particularly important when running on Windows. While Python's garbage collector should eventually clean up the resources for you, closing them makes them available earlier.

However, if you close a clip too early, methods on the clip (and any clips derived from it) become unsafe.

So, the rules of thumb are:

Skip to main content

> - Call `close()` on any clip that you **construct** once you have finished using it and have also finished using any clip that was derived from it.
> - Even if you close a `CompositeVideoClip` instance, you still need to close the clips it was created from.
> - Otherwise, if you have a clip that was created by deriving it from from another clip (e.g. by calling `with_mask()`), then generally you shouldn't close it. Closing the original clip will also close the copy.

Clips act as context managers. This means you can use them with a `with` statement, and they will automatically be closed at the end of the block, even if there is an exception.

```
from moviepy import *

# clip.close() is implicitly called, so the lock on my_audiofile.mp3 file
# is immediately released.
try:
    with AudioFileClip("example.wav") as clip:
        raise Exception("Let's simulate an exception")
except Exception as e:
    print("{}".format(e))
```

Categories of video clips

Video clips are the building blocks of longer videos. Technically, they are clips with a `clip.get_frame(t)` method which outputs a `HxWx3` numpy array representing the frame of the clip at time `t`.

There are two main type of video clips:

- animated clips (made with `VideoFileClip`, `VideoClip` and `ImageSequenceClip`), which will always have duration.
- unanimated clips (made with `ImageClip`, `VideoClip`\`TextClip` and `ColorClip`), which show the same picture for an a-priori infinite duration.

There are also special video clips called masks, which belong to the categories above but output greyscale frames indicating which parts of another clip are visible or not.

Skip to main content

A video clip can carry around an audio clip (`AudioClip`) in `audio` which is its *soundtrack*, and a mask clip in `mask`.

Animated clips

These are clips whose image will change over time, and which have a duration and a number of Frames Per Second.

VideoClip

`VideoClip` is the base class for all the other video clips in MoviePy. If all you want is to edit video files, you will never need it. This class is practical when you want to make animations from frames that are generated by another library. All you need is to define a function `frame_function(t)` which returns a *HxWx3* numpy array (of 8-bits integers) representing the frame at time `t`.

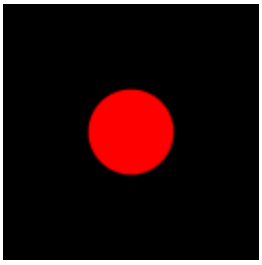
Here is an example where we will create a pulsating red circle with graphical library pillow.

```
from PIL import Image, ImageDraw
import numpy as np
from moviepy import VideoClip
import math

WIDTH, HEIGHT = (128, 128)
RED = (255, 0, 0)


def frame_function(t):
    frequency = 1  # One pulse per second
    coef = 0.5 * (1 + math.sin(2 * math.pi * frequency * t))  # radius varies over
    radius = WIDTH * coef

    x1 = WIDTH / 2 - radius / 2
    y1 = HEIGHT / 2 - radius / 2
    x2 = WIDTH / 2 + radius / 2
    y2 = HEIGHT / 2 + radius / 2

    img = Image.new("RGB", (WIDTH, HEIGHT))
    draw = ImageDraw.Draw(img)
    draw.ellipse((x1, y1, x2, y2), fill=RED)

    return np.array(img)  # returns a 8-bit RGB array


# we define a 2s duration for the clip to be able to render it later
```

Skip to main content

```
# we must set a framerate because VideoClip have no framerate by default
clip.write_gif("circle.gif", fps=15)
```

Resulting in this.



Note

Clips that are made with a `frame_function` do not have an explicit frame rate nor duration by default, so you must provide duration at clip creation and a frame rate (`fps`, frames per second) for **`write_gif()`** and **`write_videofile()`**, and more generally for any methods that requires iterating through the frames.

For more, see **`VideoClip`**.

VideoFileClip

A **`VideoFileClip`** is a clip read from a video file (most formats are supported) or a GIF file. This is probably one of the most used object ! You load the video as follows:

```
from moviepy import VideoFileClip

myclip = VideoFileClip("example.mp4")

# video file clips already have fps and duration
print("Clip duration: {}".format(myclip.duration))
print("Clip fps: {}".format(myclip.fps))

myclip = myclip.subclipped(0.5, 2)  # Cutting the clip between 0.5 and 2 secs.
print("Clip duration: {}".format(myclip.duration))  # Cuting will update duration
print("Clip fps: {}".format(myclip.fps))  # and keep fps
# the output video will be 1.5 sec long and use original fps
myclip.write_videofile("result.mp4")
```

Skip to main content

> **Note**
>
> These clips will have an `fps` (frame per second) and `duration` attributes, which will be transmitted if you do small modifications of the clip, and will be used by default in `write_gif()`, `write_videofile()`, etc.

For more, see `VideoFileClip`.

ImageSequenceClip

This `ImageSequenceClip` is a clip made from a series of images :

```
from moviepy import ImageSequenceClip

# A clip with a list of images showed for 1 second each
myclip = ImageSequenceClip(
    [
        "example_img_dir/image_0001.jpg",
        "example_img_dir/image_0002.jpg",
        "example_img_dir/image_0003.jpg",
    ],
    durations=[1, 1, 1],
)
# 3 images, 1 seconds each, duration = 3
print("Clip duration: {}".format(myclip.duration))
# 3 seconds, 3 images, fps is 3/3 = 1
print("Clip fps: {}".format(myclip.fps))

# This time we will load all images in the dir, and instead of showing theme
# for X seconds, we will define FPS
myclip2 = ImageSequenceClip("./example_img_dir", fps=30)
# fps = 30, so duration = nb images in dir / 30
print("Clip duration: {}".format(myclip2.duration))
print("Clip fps: {}".format(myclip2.fps))  # fps = 30

# the gif will be 30 fps, its duration will depend on the number of
# images in dir
myclip.write_gif("result.gif")  # the gif will be 3 sec and 1 fps
myclip2.write_gif("result2.gif")
```

When creating an image sequence, `sequence` can be either a list of image names (that will be *played* in the provided order), a folder name (played in alphanumerical order), or a list of frames (Numpy arrays), obtained for instance from other clips.

Skip to main content

> **Warning**
>
> All the images in list/folder/frames must be of the same size, or an exception will be raised

For more, see `ImageSequenceClip`.

DataVideoClip

`DataVideoClip` is a video clip who take a list of datasets, a callback function, and make each frame by iterating over dataset and invoking the callback function with the current data as first argument.

You will probably never use this. But if you do, think of it like a `VideoClip`, where you make frames not based on time, but based on each entry of a data list.

```
"""Let's make a clip where frames depend on values in a list"""

from moviepy import DataVideoClip
import numpy as np

# Dataset will just be a list of colors as RGB
dataset = [
    (255, 0, 0),
    (0, 255, 0),
    (0, 0, 255),
    (0, 255, 255),
    (255, 0, 255),
    (255, 255, 0),
]


# The function make_frame take data and create an image of 200x100 px
# filled with the color given in the dataset
def frame_function(data):
    frame = np.full((100, 200, 3), data, dtype=np.uint8)
    return frame


# We create the DataVideoClip, and we set FPS at 2, making a 3s clip
# (because len(dataset) = 6, so 6/2=3)
myclip = DataVideoClip(data=dataset, data_to_frame=frame_function, fps=2)

# Modifying fps here will change video FPS, not clip FPS
myclip.write_videofile("result.mp4", fps=30)
```

Skip to main content

For more, see For more, see `DataVideoClip`.

UpdatedVideoClip

> **Warning**
>
> This is really advanced usage, you will probably never need it, if you do, please go read the code.

`UpdatedVideoClip` is a video whose frame_function requires some objects to be updated before we can compute it.

This is particularly practical in science where some algorithm needs to make some steps before a new frame can be generated, or maybe when trying to make a video based on a live exterior context.

When you use this, you pass a world object to it. A world object is an object who respect these 3 rules:

1. It has a `clip_t` property, indicating the current world time.
2. It has an `update()` method, that will update the world state and is responsible for increasing `clip_t` when a new frame can be drown.
3. It has a `to_frame()` method, that will render a frame based on world current state.

On `get_frame()` call, your `UpdatedVideoClip` will try to update the world until `world.clip_t` is superior or equal to frame time, then it will call `world.to_frame()`.

```
from moviepy import UpdatedVideoClip
import numpy as np
import random


class CoinFlipWorld:
    """A simulation of coin flipping.

    Imagine we want to make a video that become more and more red as we repeat sam
    on coinflip in a row because coinflip are done in real time, we need to wait
    until a winning row is done to be able to make the next frame.
    This is a world simulating that. Sorry, it's hard to come up with examples..."
```

Skip to main content

```
        FPS is usefull because we must increment clip_t by 1/FPS to have
        UpdatedVideoClip run with a certain FPS
        """
        self.clip_t = 0
        self.win_strike = 0
        self.reset = False
        self.fps = fps

    def update(self):
        if self.reset:
            self.win_strike = 0
            self.reset = False

        print("strike : {}, clip_t : {}".format(self.win_strike, self.clip_t))
        print(self.win_strike)

        # 0 tails, 1 heads, this is our simulation of coinflip
        choice = random.randint(0, 1)
        face = random.randint(0, 1)

        # We win, we increment our serie and retry
        if choice == face:
            self.win_strike += 1
            return

        # Different face, we increment clip_t and set reset so we will reset on ne
        # We don't reset immediately because we will need current state to make fr
        self.reset = True
        self.clip_t += 1 / self.fps

    def to_frame(self):
        """Return a frame of a 200x100 image with red more or less intense based
        on number of victories in a row."""
        red_intensity = 255 * (self.win_strike / 10)
        red_intensity = min(red_intensity, 255)

        # A 200x100 image with red more or less intense based on number of victori
        return np.full((100, 200, 3), (red_intensity, 0, 0), dtype=np.uint8)


world = CoinFlipWorld(fps=5)

myclip = UpdatedVideoClip(world=world, duration=10)
# We will set FPS to same as world, if we was to use a different FPS,
# the lowest from world.fps and our write_videofile fps param
# will be the real visible fps
myclip.write_videofile("result.mp4", fps=5)
```

Unanimated clips

Skip to main content

These are clips whose image will, at least before modifications, stay the same. By default they have no duration nor FPS, meaning you will need to define them before doing operations needing such information (for example, rendering).

ImageClip

`ImageClip` is the base class for all unanimated clips, it's a video clip that always displays the same image. Along with `VideoFileClip` it's one of the most used kind of clip. You can create one as follows:

```
"""Here's how you transform a VideoClip into an ImageClip from an image, from
arbitrary data, or by extracting a frame at a given time"""

from moviepy import ImageClip, VideoFileClip
import numpy as np

# Random RGB noise image of 200x100
noise_image = np.random.randint(low=0, high=255, size=(100, 200, 3))

myclip1 = ImageClip("example.png")  # You can create it from a path
myclip2 = ImageClip(noise_image)  # from a (height x width x 3) RGB numpy array
# Or load videoclip and extract frame at a given time
myclip3 = VideoFileClip("./example.mp4").to_ImageClip(t="00:00:01")
```

For more, see `ImageClip`.

TextClip

A `TextClip` is a clip that will turn a text string into an image clip.

`TextClip` accept many parameters, letting you configure the appearance of the text, such as font and font size, color, interlining, text alignment, etc.

The font you want to use must be an OpenType font, and you will set it by passing the path to the font file.

Here are a few example of using `TextClip` :

```
from moviepy import TextClip
```

Skip to main content

```
# First we use as string and let system autocalculate clip dimensions to fit the t
# we set clip duration to 2 secs, if we do not, it got an infinite duration
txt_clip1 = TextClip(
    font=font,
    text="Hello World !",
    font_size=30,
    color="#FF0000",  # Red
    bg_color="#FFFFFF",
    duration=2,
)
# This time we load text from a file, we set a fixed size for clip and let the sys
# allowing for line breaking
txt_clip2 = TextClip(
    font=font,
    filename="./example.txt",
    size=(500, 200),
    bg_color="#FFFFFF",
    method="caption",
    color=(0, 0, 255, 127),
)  # Blue with 50% transparency

# we set duration, because by default image clip are infinite, and we cannot rende
txt_clip2 = txt_clip2.with_duration(2)
# ImageClip have no FPS either, so we must defined it
txt_clip1.write_videofile("result1.mp4", fps=24)
txt_clip2.write_videofile("result2.mp4", fps=24)
```

Note

The parameter `method` let you define if text should be written and overflow if too long (`label`) or be automatically broken over multiple lines (`caption`).

For a more detailed explanation of all the parameters, see **`TextClip`**.

ColorClip

A **`ColorClip`** is a clip that will return an image of only one color. It is sometimes useful when doing compositing (see Compositing multiple clips).

```
from moviepy import ColorClip

# Color is passed as a RGB tuple
myclip = ColorClip(size=(200, 100), color=(255, 0, 0), duration=1)
# We really don't need more than 1 fps do we ?
myclip.write_videofile("result.mp4", fps=1)
```

Skip to main content

For more, see `ColorClip`.

Mask clips

Masks are a special kind of `VideoClip` with the property `is_mask` set to `True`. They can be attached to any other kind of `VideoClip` through method `with_mask()`.

When a clip as a mask attached to it, this mask will indicate which pixels will be visible when the clip is composed with other clips (see Compositing multiple clips). Masks are also used to define transparency when you export the clip as GIF file or as a PNG.

The fundamental difference between masks and standard clips is that standard clips output frames with 3 components (R-G-B) per pixel, comprised between 0 and 255, while a mask has just one component per pixel, between 0 and 1 (1 indicating a fully visible pixel and 0 a transparent pixel). Seen otherwise, a mask is always in greyscale.

When you create or load a clip that you will use as a mask you need to declare it. You can then attach it to a clip with the same dimensions :

```
from moviepy import VideoClip, ImageClip, VideoFileClip
import numpy as np

# Random RGB noise image of 200x100
frame_function = lambda t: np.random.rand(100, 200)

# To define the VideoClip as a mask, just pass parameter is_mask as True
maskclip1 = VideoClip(frame_function, duration=4, is_mask=True)  # A random noise
maskclip2 = ImageClip("example_mask.jpg", is_mask=True)  # A fixed mask as jpeg
maskclip3 = VideoFileClip("example_mask.mp4", is_mask=True)  # A video as a mask

# Load our basic clip, resize to 200x100 and apply each mask
clip = VideoFileClip("example.mp4")
clip_masked1 = clip.with_mask(maskclip1)
clip_masked2 = clip.with_mask(maskclip2)
clip_masked3 = clip.with_mask(maskclip3)
```

Skip to main content

> **Note**
>
> In the case of video and image files, if these are not already black and white they will be converted automatically.
>
> Also, when you load an image with an *alpha layer*, like a PNG, MoviePy will use this layer as a mask unless you pass `transparent=False`.

Any video clip can be turned into a mask with **`to_mask()`**, and a mask can be turned to a standard RGB video clip with **`to_RGB()`**.

Masks are treated differently by many methods (because their frames are different) but at the core, they are **`VideoClip`**, so you can do with theme everything you can do with a video clip: modify, cut, apply effects, save, etc.

Using audio elements with audio clips

In addition to **`VideoClip`** for visual, you can use audio elements, like an audio file, using the **`AudioClip`** class.

Both are quite similar, except **`AudioClip`** method **`get_frame()`** return a numpy array of size `Nx1` for mono, and size `Nx2` for stereo.

AudioClip

`AudioClip` is the base class for all audio clips. If all you want is to edit audio files, you will never need it.

All you need is to define a function `frame_function(t)` which returns a `Nx1` or `Nx2` numpy array representing the sound at time `t`.

```
from moviepy import AudioClip
import numpy as np


def audio_frame(t):
    """Producing a sinewave of 440 Hz -> note A"""
```

Skip to main content

```
audio_clip = AudioClip(frame_function=audio_frame, duration=3)
```

For more, see `AudioClip`.

AudioFileClip

`AudioFileClip` is used to load an audio file. This is probably the only kind of audio clip you will use.

You simply pass it the file you want to load :

```
from moviepy import *

# Works for audio files, but also videos file where you only want the keep the aud
clip = AudioFileClip("example.wav")
clip.write_audiofile("./result.wav")
```

For more, see `AudioFileClip`.

AudioArrayClip

`AudioArrayClip` is used to turn an array representing a sound into an audio clip. You will probably never use it, unless you need to use the result of some third library without using a temporary file.

You need to provide a numpy array representing the sound (of size `Nx1` for mono, `Nx2` for stereo), and the number of fps, indicating the speed at which the sound is supposed to be played.

```
"""Let's create an audioclip from values in a numpy array."""

import numpy as np
from moviepy import AudioArrayClip

# We want to play these notes
notes = {"A": 440, "B": 494, "C": 523, "D": 587, "E": 659, "F": 698}

note_duration = 0.5
total_duration = len(notes) * note_duration
sample_rate = 44100  # Number of samples per second
```

Skip to main content

```
n_frames = note_size * len(notes)


def frame_function(t, note_frequency):
    return np.sin(note_frequency * 2 * np.pi * t)


# At this point one could use this audioclip which generates the audio on the fly
# clip = AudioFileClip(frame_function)

# We generate all frames timepoints
```