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

Previewing and saving video clips

Once you are down working with your clips, the last step will be to export the result into a video/image file, or sometimes to simply preview it in order to verify everything is working as expected.

Previewing a clip

When you are working with a clip, you will frequently need to have a peak at what your clip looks like, either to verify that everything is working as intended, or to check how things looks.

To do so you could render your entire clip into a file, but that's a pretty long task, and you only need a quick look, so a better solution exists: previewing.

Preview a clip as a video

> **Warning**
>
> You must have `ffplay` installed and accessible to MoviePy to be able to use `preview()`. If you'r not sure, take a look Installation of additional binaries

The first thing you can do is to preview your clip as a video, by calling method `preview()` on your clip:

```
from moviepy import *

myclip = VideoFileClip("./example.mp4").subclipped(0, 1)  # Keep only 0 to 1 sec
```

Skip to main content

```
myclip.preview()

# We preview our clip as video, but with a custom FPS for video and audio
# making it less consuming for our computer
myclip.preview(fps=5, audio_fps=11000)

# Now we preview without audio
myclip.preview(audio=False)
```

You will probably frequently want to preview only a small portion of your clip, though `preview` do not offer such capabilities, you can easily emulate such behavior by using **`subclipped()`**.

> **Note**
>
> It is quite frequent for a clip preview to be out of sync, or to play slower than it should. It means that your computer is not powerful enough to render the clip in real time.
>
> Don't hesitate to play with the options of preview: for instance, lower the fps of the sound (11000 Hz is still fine) and the video. Also, downsizing your video with resize can help.

For more info, see **`preview()`**.

> **Note**
>
> A quite similar function is also available for **`AudioClip()`**, see **`ffplay_audiopreview()`**.

Preview just one frame of a clip

In a lot of situation, you don't really need to preview your all clip, seeing only one frame is enough to see how it looks like and to make sure everything goes as expected.

To do so, you can use the method **`show()`** on your clip, passing the frame time as an argument:

```
from moviepy import *

myclip = VideoFileClip("./example.mp4")
```

Skip to main content

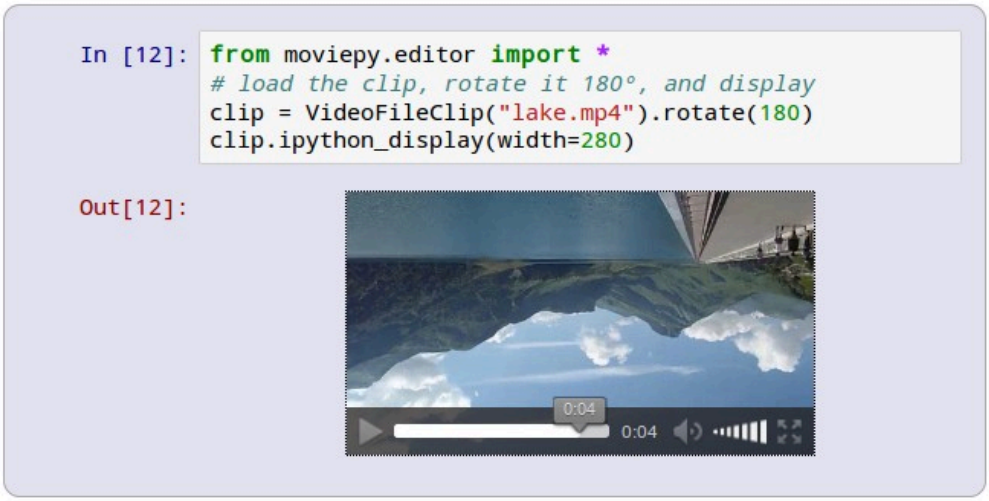
```
# We show the frame at point 00:00:01.5 of our clip
myclip.show(1.5)

# We want to see our clip without applying his mask
myclip.show(1.5, with_mask=False)
```

Contrary to video previewing, show does not require `ffplay`, but use `pillow` `Image.show` function.

For more info, see **`show()`**.

Showing a clip in Jupyter Notebook

If you work with a Jupyter Notebook, it can be very practical to display your clip the notebook. To do so, you can use the method **`display_in_notebook()`** on your clip.



With **`display_in_notebook()`** you can embed videos, images and sounds, either from a file or directly from a clip:

```
from moviepy import *

# ...
# ... some jupyter specifics stuff
# ...

my_video_clip = VideoFileClip("./example.mp4")
my_image_clip = ImageClip("./example.png")
my_audio_clip = AudioFileClip("./example.wav")

# We can show any type of clip
```

Skip to main content

```
my_audio_clip.display_in_notebook()  # embeds a sound

# We can display only a snaphot of a video
my_video_clip.display_in_notebook(t=1)

# We can provide any valid HTML5 option as keyword argument
# For instance, if the clip is too big, we can set width
my_video_clip.display_in_notebook(width=400)

# We can also make it loop, for example to check if a GIF is
# looping as expected
my_video_clip.display_in_notebook(autoplay=1, loop=1)
```

Warning

Know that `display_in_notebook()` will only work if it is on the last line a the notebook cell.

Also, note that `display_in_notebook()` actually embeds the clips physically in your notebook. The advantage is that you can move the notebook or put it online and the videos will work. The drawback is that the file size of the notebook can become very large. Depending on your browser, re-computing and displaying at video many times can take some place in the cache and the RAM (it will only be a problem for intensive uses). Restarting your browser solves the problem.

For more info, see `display_in_notebook()`.

Save your clip into a file

Once you are satisfied with how your clip looks, you can save it into a file, a step known in video edition as rendering. MoviePy offer various way to save your clip.

Video files (.mp4, .webm, .ogv...)

The obvious first choice will be to write your clip to a video file, which you can do with `write_videofile()`:

```
from moviepy import *
```

Skip to main content

```
background = VideoFileClip("long_examples/example2.mp4").subclipped(0, 10)
title = TextClip(
    "./example.ttf",
    text="Big Buck Bunny",
    font_size=80,
    color="#fff",
    text_align="center",
    duration=3,
).with_position(("center", "center"))

# We make our final clip through composition
final_clip = CompositeVideoClip([background, title])

# And finally we can write the result into a file

# Here we just save as MP4, inheriting FPS, etc. from final_clip
final_clip.write_videofile("result.mp4")

# Here we save as MP4, but we set the FPS of the clip to our own, here 24 fps, lik
final_clip.write_videofile("result24fps.mp4", fps=24)

# Now we save as WEBM instead, and we want tu use codec libvpx-vp9 (usefull when m
# We also want ffmpeg compression optimisation as minimal as possible. This will n
# the video quality and it will decrease time for encoding, but increase final fil
# Finally, we want ffmpeg to use 4 threads for video encoding. You should probably
# to default, as ffmpeg is already quite good at using the best setting on his own
final_clip.write_videofile(
    "result.webm", codec="libvpx-vp9", fps=24, preset="ultrafast", threads=4
)
```

MoviePy can find the a default codec name for the most common file extensions. If you want to use exotic formats or if you are not happy with the defaults you can provide the codec with `codec='mpeg4'` for instance.

There are many many options when you are writing a video (bitrate, parameters of the audio writing, file size optimization, number of processors to use, etc.), and we will not go in details into each. So, for more info, see **`write_videofile()`**.

Skip to main content

> **Note**
>
> Though you are encouraged to play with settings of `write_videofile`, know that lowering the optimization preset or increasing the number of threads will not necessarily improve the rendering time, as the bottleneck may be on MoviePy computation of each frame and not in ffmpeg encoding.
>
> Also, know that it is possible to pass additional parameters to ffmpeg command line invoked by MoviePy by using the `ffmpeg_params` argument.

Sometimes it is impossible for MoviePy to guess the `duration` attribute of the clip (keep in mind that some clips, like ImageClips displaying a picture, have *a priori* an infinite duration). Then, the `duration` must be set manually with **`with_duration()`**:

```
from moviepy import *

# By default an ImageClip has no duration
my_clip = ImageClip("example.png")

try:
    # This will fail! We cannot write a clip with no duration!
    my_clip.write_videofile("result.mp4")
except:
    print("Cannot write a video without duration")

# By calling with_duration on our clip, we fix the problem! We also need to set fp
my_clip.with_duration(2).write_videofile("result.mp4", fps=1)
```

> **Note**
>
> A quite similar function is also available for **`AudioClip()`**, see **`write_audiofile()`**.

Export a single frame of the clip

As for previewing, sometimes you will need to export only one frame of a clip, for example to create the preview image of a video. You can do so with **`save_frame()`**:

```
from moviepy import *
```

Skip to main content

```
myclip = VideoFileClip("example.mp4")
myclip.save_frame("result.png", t=1)  # Save frame at 1 sec
```

For more info, see `save_frame()`.

Animated GIFs

In addition to writing video files, MoviePy also let you write GIF file with `write_gif()`:

```
from moviepy import *

myclip = VideoFileClip("example.mp4").subclipped(0, 2)

# Here we just save as GIF
myclip.write_gif("result.gif")

# Here we save as GIF, but we set the FPS of our GIF at 10
myclip.write_gif("result.gif", fps=10)
```

For more info, see `write_gif()`.

Export all the clip as images in a directory

Lastly, you may wish to export an entire clip as an image sequence (multiple images in one directory, one image per frame). You can do so with the function `write_images_sequence()`:

```
from moviepy import *
import os

myclip = VideoFileClip("example.mp4")
```