Assignment 6b

1. Names and Emails

Boyang Wang (boyangwa, boyangwa@umich.edu)
Yichen Zhong (yichenzh, yichenzh@umich.edu)

2. Selected Project

In HW6A, we selected the MoviePy project. MoviePy is a python module implementation based on FFMPEG for video development. We take it as our target contribution project. The official GitHub page for this project is https://github.com/Zulko/moviepy. Specifically, this video editing module can be used for basic operations (like cuts, concatenations, and title insertions), video compositing (a.k.a. non-linear editing), video processing, or to create advanced effects. Its backend is based on the FFMPEG (https://ffmpeg.org/) which is the most popular video processing tool in this field. Above FFMPEG, its whole implementation is based on pure Python (not a multi-language project). After we got the hw6a report feedback, we unluckily found that there was a problem without task selection. Originally, it said that it should come from Github issues but we choose one that is from stack overflow. In order to compensate for this homework, we choose two more GitHub issues presented by others to do even though we almost finished task 1 in hw6a. These two GitHub issues are all from MoviePy and they are related to the same file (Clip.py).

3. Social Goal Indication

MoviePy is one of the few open-source software packages focusing on video processing development in the python library family. The users who use this module are mainly professional audio and video developers and some personal developers who want to quickly analyze or edit video information, such as video resolution, frames per second, video duration, etc. Our task focuses on improving the existing functions of MoviePy so that developers can define the parameters of output video files more personally when using this package. This saves video developers time in the research and development process and facilitates them to use MoviePy better to build their products. Our mission does contribute to the open-source community to some extent, which bodes well for the programming community.

4. Project Context

MoviePy aims to solve the problem of limited open-source software in video processing and research and development. MoviePy was developed by a GitHub user named Zulko in Cambridge, MA. The project was established on August 11, 2013, and has mainly 9 stable maintainers. The existence of this project fills the void of open-source software for video processing. FFMPEG is also a software for audio, video, and image processing, but MoviePy has made many innovative functions for video development based on borrowing the advantages of FFMPEG. From this point of view, MoviePy is more focused on video processing and has been in It is far more functional than FFMPEG. On the other hand, CV2 can also constitute competition and can also extract video frames for analysis in terms of video processing. However, CV2 can only support sequential extraction of frames from beginning to end for analysis, while MoviePy can support entry from any node and support random extraction. MoviePy has many functions that other similar software does not have in video processing and is a very important open-source software.

5. Project Governance

The code review and acceptance process are similar to the informal way of passing code. Code contributors pull requests to the project's main branch and create a new branch for submission. After the automated test suite has run through the uploaded code, the MoviePy maintainers review the code and make a decision to accept or reject the change. Here are the specifics:

Communication:

- Communication is mainly based on Github. Coordination between contributors is primarily through Keep messages on GitHub issues and pull requests on-topic and to the point. Each comment triggers a notification, which is sent to several maintainers.
- For longer or more in-depth discussions, use MoviePy Gitter, a discussion platform. If these discussions lead to a decision, like a merge/reject, the developer shall leave a message on the relevant MoviePy issue to document the outcome of the discussion/the reason for the decision.
- There is also an email librelist moviepy@librelist.com to connect with maintainers.
- There is also a Reddit forum called MoviePy - Pythonic movie editing to let users ask for detailed functionalities and suggest improvements.

Development Expectations and Requirements

- Before development: developers shall fork the official MoviePy repository to their own GitHub account, clone the repository to their local machine, add the official MoviePy repository as a second remote with an alias upstream, install a virtual environment with all dependencies included, and configure pre-commit hooks running.
- During development: developers shall keep their local master branch up-to-date with the official repo's master by periodically fetching/pulling it, and remember only to make

changes in their develop branches. Developers should also base any changes submitted on the most recent master.
- After development: developers should run the test suite(pytest) and push their local development branch into their GitHub fork before submitting pull requests. After submitting pull requests, developers must fill out a template describing changes.

Quality Insurance

- Developers should respect PEP8 conventions.
- If developers introduce new functionality or fix a bug, they shall document it in the docstring or with code comments.
- MoviePy's team adopted pre-commit to run code checks using black, flake8, and isort, after developers submit their code changes request.

6. Task Description

Task 1: (Same as HW 6a) Modify decoded frames' output resolution to developers' target output size.

After using the API of MoviePy in daily life, we found that the "iter_frames()" API cannot customize the output resolution size directly by developers. Even though we can DIY the fps (frames per second) needed from this API, changing output size options are not provided. Instead of implementing it by developers themselves, it will be very convenient if we can do so by changing the parameters of the API, like this "iter_frames(output=(target_width, target_height))". Then, developers don't need to care about how to implement the complicated resize algorithm themselves. (There are too many kinds of resize algorithms from a computer vision perspective and most developers are reluctant to know the algorithm or implementation details of them all during the utilization of MoviePy) With such a feature, the utilization of MoviePy will be more concise and functional. Meanwhile, such requests also occur in stack overflow as our 6a showed. (**For GitHub issues task, it will be task2 and 3**) The implementation may not be too complex, but it is very meaningful for developers who use MoviePy often.

Task 2: Extra frames at the beginning and dropped frames at the end write_videofile

This is a new task that we want to do besides task 1. This is a GitHub issue and the link is https://github.com/Zulko/moviepy/issues/1831. One user wants to extract 2-second sub-clips from a 100 fps video and concatenate them in a new video using a combination of sub-clip, concatenate_videofile, and write_videofile. The expected behavior is that there should be a

video composed of 200 frame sub-clips precisely centered around supplied time points. But actually, the First sub-clip has a length of 210 frames (the first ten frames are repeated twice) and the last sub-clip has 190. All the middle ones are 200 frames in length with the supplied timepoint in position accurately centered at 100 frames from the sub-clip start.
We're guessing this could be due to I-frame or GOP (group of pictures) settings in the video codec. Sometimes, when we want to extract a video clip via FFmpeg, MoviePy may take several frame adjustments (maybe more or fewer frames are cut) while cutting the clip from the video. Typically, their adjustments are directly related to video encoding on groups of pictures (GOP). However, when we tried to reproduce this bug, we tested a lot of data (video input that strictly matches the 100fps environment) but found no related errors presented in the GitHub issues. Thus, we tried to contact the person who reported the bug and asked him to give more information about relevant parameters and even input video examples (as presented at the bottom of the link provided above). Hopefully, this may be a false bug and we found it. We want the maintainer to close this issue in the near future, such that it won't distract other developers anymore if it is a false bug.

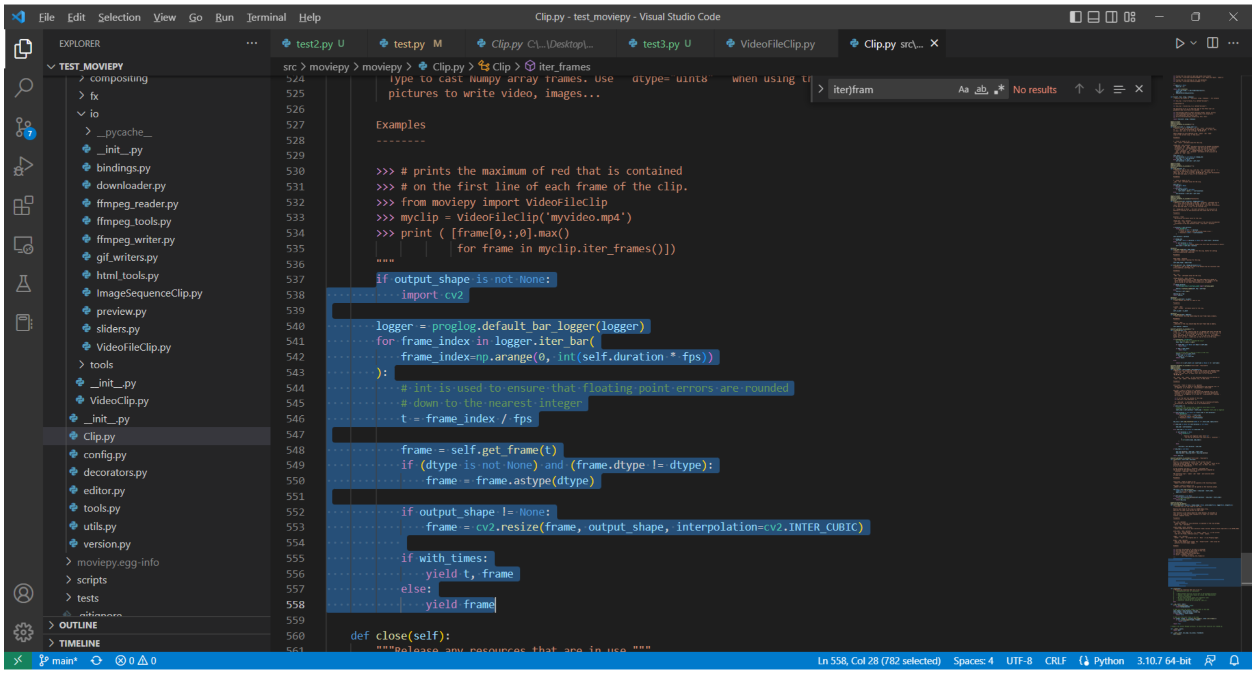
Task 3: Moviepy processes video 1 sec ahead of it

This is also a new task that we want to do besides hw6a. This is also a GitHub issue with the link https://github.com/Zulko/moviepy/issues/1816. While cutting a video by using the set_duration method in MoviePy, the user specified the seconds but the processed video he got is one second ahead of what he gave in the parameters. When we went to reproduce the bug, we found that the duration variable of set_duration would have an out-of-boundary bug, so we imitated the end time such that it must be less than the whole duration. If it is more than the whole duration, it will raise an error in Python format. Without such a mechanism we want to propose, if developers set an end time out of the boundary, the program will raise a lot of error messages in the terminal and the clip video may not be run correctly, which makes the code messy and stops the program unexpectedly later after this line. Thus, we have to make a raise error mechanism when the maximum duration of this video is exceeded to protect the code from invalid overflow access.

7. Submitted Artifacts

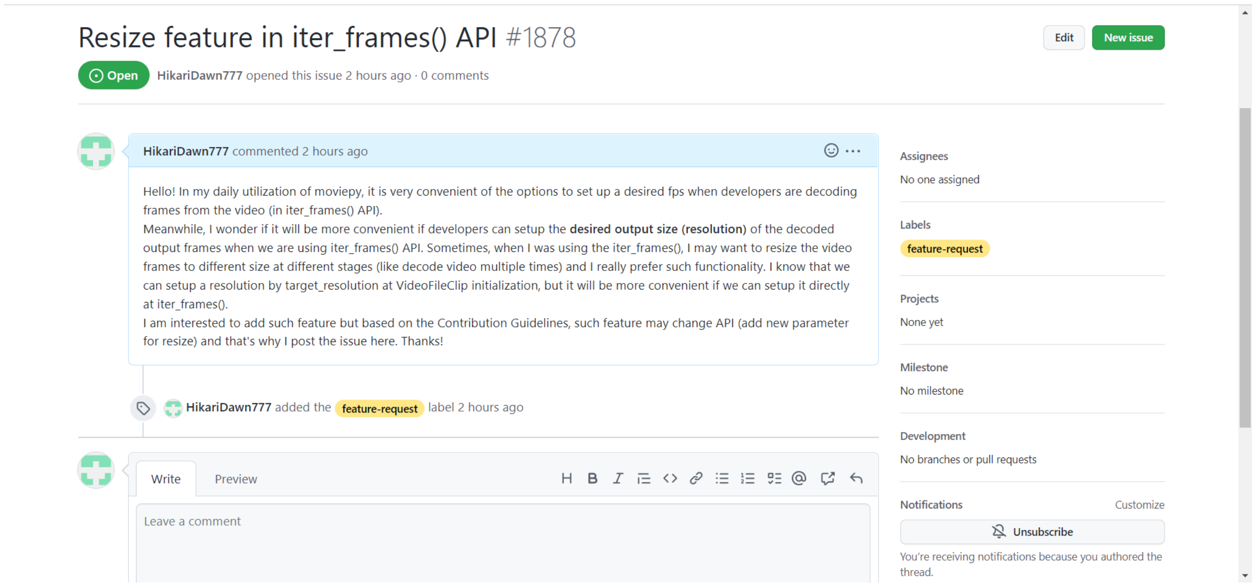
Task1:

We just want to be clarified here.We finished all the functionality of it and exhibit it here.



```
        Type to cast Numpy array frames. Use  dtype="uint8"  when using th
        pictures to write video, images...

        Examples
        --------

        >>> # prints the maximum of red that is contained
        >>> # on the first line of each frame of the clip.
        >>> from moviepy import VideoFileClip
        >>> myclip = VideoFileClip('myvideo.mp4')
        >>> print ( [frame[0,:,0].max()
                         for frame in myclip.iter_frames()])
        """
        if output_shape is not None:
            import cv2

        logger = proglog.default_bar_logger(logger)
        for frame_index in logger.iter_bar(
            frame_index=np.arange(0, int(self.duration * fps))
        ):
            # int is used to ensure that floating point errors are rounded
            # down to the nearest integer
            t = frame_index / fps

            frame = self.get_frame(t)
            if (dtype is not None) and (frame.dtype != dtype):
                frame = frame.astype(dtype)

            if output_shape != None:
                frame = cv2.resize(frame, output_shape, interpolation=cv2.INTER_CUBIC)

            if with_times:
                yield t, frame
            else:
                yield frame

    def close(self):
        """Release any resources that are in use."""
```

We want to push it at first, but based on the pull request requirement mentioned here (https://github.com/Zulko/moviepy/blob/master/CONTRIBUTING.md), we cannot directly pull request it because we change parameters in the “iter_frames” API and we have to **first publish a GitHub issue** of a pull request for it (based on the contribution requirement). And the following screenshot is the pull request we have done:
The link is here also: https://github.com/Zulko/moviepy/issues/1878



Resize feature in iter_frames() API #1878

Open HikariDawn777 opened this issue 2 hours ago · 0 comments

HikariDawn777 commented 2 hours ago

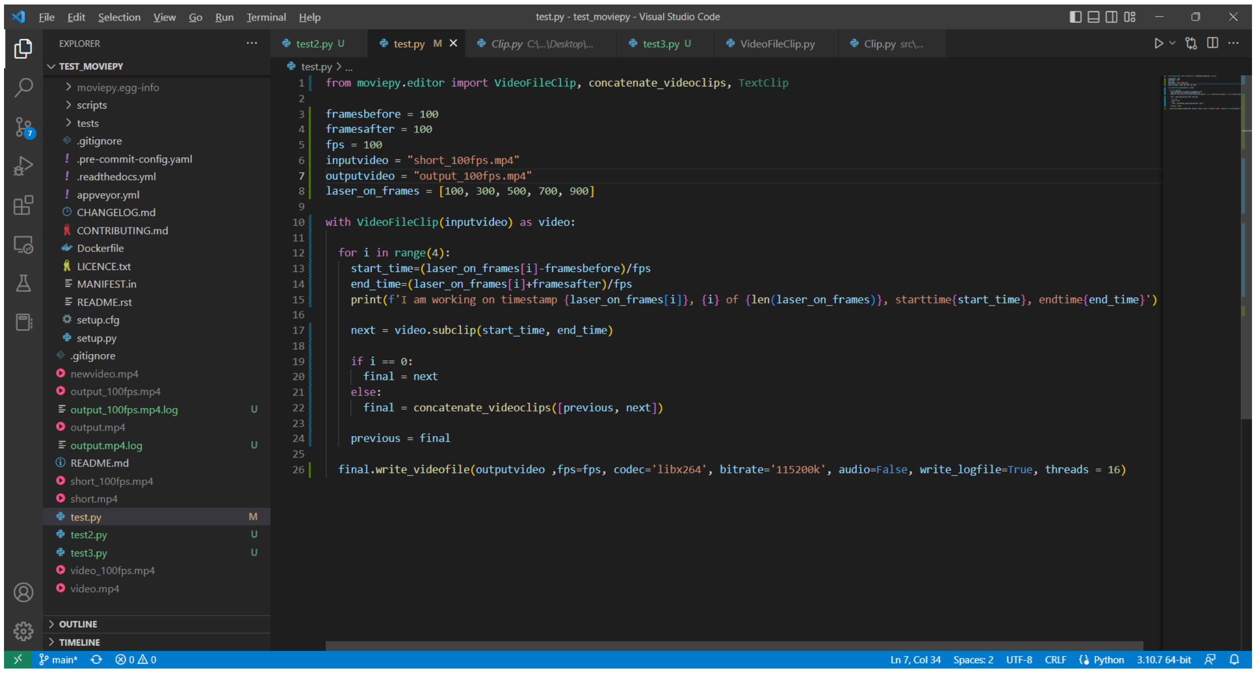
Hello! In my daily utilization of moviepy, it is very convenient of the options to set up a desired fps when developers are decoding frames from the video (in iter_frames() API).
Meanwhile, I wonder if it will be more convenient if developers can setup the **desired output size (resolution)** of the decoded output frames when we are using iter_frames() API. Sometimes, when I was using the iter_frames(), I may want to resize the video frames to different size at different stages (like decode video multiple times) and I really prefer such functionality. I know that we can setup a resolution by target_resolution at VideoFileClip initialization, but it will be more convenient if we can setup it directly at iter_frames().
I am interested to add such feature but based on the Contribution Guidelines, such feature may change API (add new parameter for resize) and that's why I post the issue here. Thanks!

HikariDawn777 added the feature-request label 2 hours ago

Assignees: No one assigned
Labels: feature-request
Projects: None yet
Milestone: No milestone
Development: No branches or pull requests
Notifications: You're receiving notifications because you authored the thread.

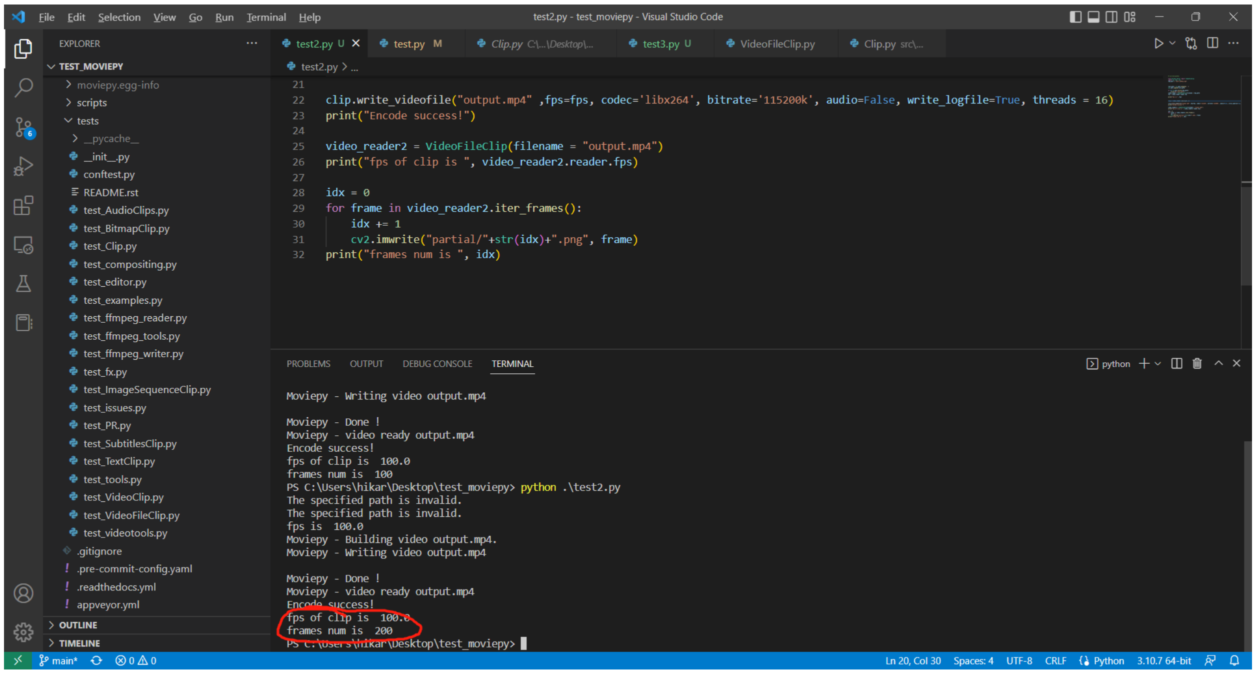
Task2:

For the second task, we tested the bugs multiple times as the following test cases. However, we cannot remake the bug again. We highly doubt the environment that the bug providers use and we think that this actually doesn't exist after hours of testing. The following (figure 7.2(b)) is the proof of the running result to support our argument. From this test case, we can see from the red circle that the output frame number is actually 200 frames instead of the 210 frames mentioned in the GitHub issues. This is the evidence of why we finally think that such GitHub issues aren't a problem.
To the best of this community (to erase unnecessary bug issues in the community), we decided to submit this question in GitHub issues (https://github.com/Zulko/moviepy/issues/1831). (I also attach a screenshot for the issues we posted on GitHub)



```
from moviepy.editor import VideoFileClip, concatenate_videoclips, TextClip

framesbefore = 100
framesafter = 100
fps = 100
inputvideo = "short_100fps.mp4"
outputvideo = "output_100fps.mp4"
laser_on_frames = [100, 300, 500, 700, 900]

with VideoFileClip(inputvideo) as video:

  for i in range(4):
    start_time=(laser_on_frames[i]-framesbefore)/fps
    end_time=(laser_on_frames[i]+framesafter)/fps
    print(f'I am working on timestamp {laser_on_frames[i]}, {i} of {len(laser_on_frames)}, starttime{start_time}, endtime{end_time}')

    next = video.subclip(start_time, end_time)

    if i == 0:
      final = next
    else:
      final = concatenate_videoclips([previous, next])

    previous = final

  final.write_videofile(outputvideo ,fps=fps, codec='libx264', bitrate='115200k', audio=False, write_logfile=True, threads = 16)
```

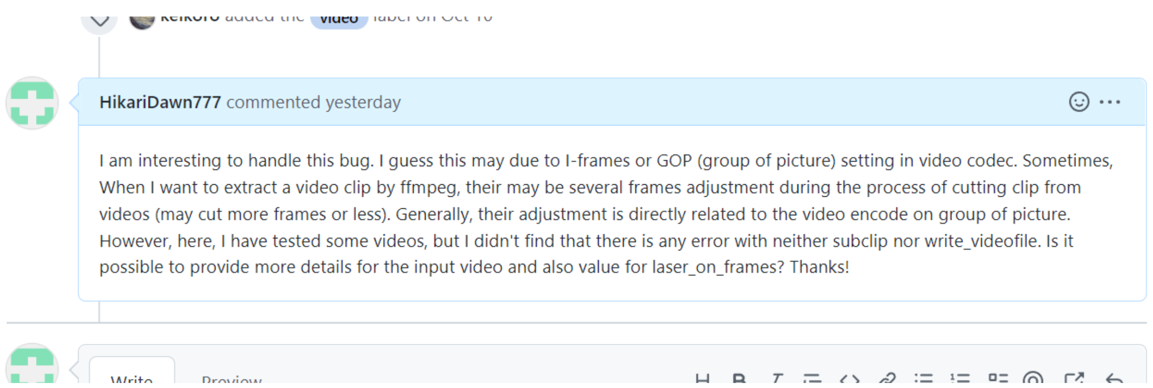
Figure 7.2(a)



```
clip.write_videofile("output.mp4" ,fps=fps, codec='libx264', bitrate='115200k', audio=False, write_logfile=True, threads = 16)
print("Encode success!")

video_reader2 = VideoFileClip(filename = "output.mp4")
print("fps of clip is ", video_reader2.reader.fps)

idx = 0
for frame in video_reader2.iter_frames():
    idx += 1
    cv2.imwrite("partial/"+str(idx)+".png", frame)
print("frames num is ", idx)
```

```
Moviepy - Writing video output.mp4

Moviepy - Done !
Moviepy - video ready output.mp4
Encode success!
fps of clip is  100.0
frames num is  100
PS C:\Users\hikar\Desktop\test_moviepy> python .\test2.py
The specified path is invalid.
The specified path is invalid.
fps is  100.0
Moviepy - Building video output.mp4.
Moviepy - Writing video output.mp4

Moviepy - Done !
Moviepy - video ready output.mp4
Encode success!
fps of clip is  100.0
frames num is  200
PS C:\Users\hikar\Desktop\test_moviepy>
```

Figure 7.2(b)



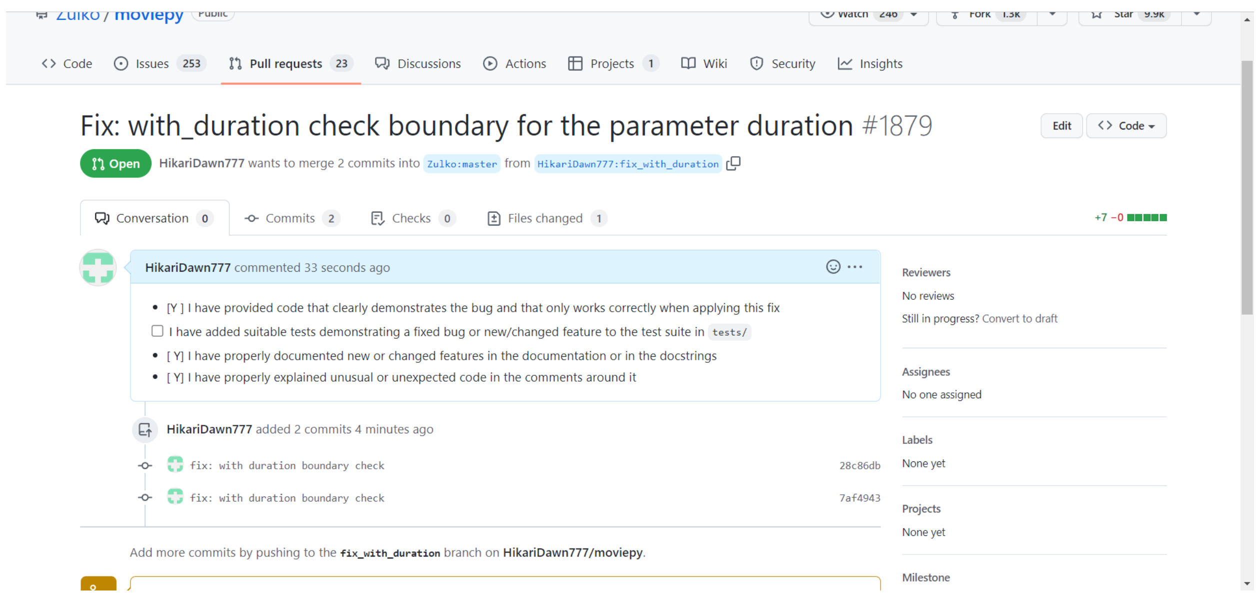
HikariDawn777 commented yesterday

I am interesting to handle this bug. I guess this may due to I-frames or GOP (group of picture) setting in video codec. Sometimes, When I want to extract a video clip by ffmpeg, their may be several frames adjustment during the process of cutting clip from videos (may cut more frames or less). Generally, their adjustment is directly related to the video encode on group of picture. However, here, I have tested some videos, but I didn't find that there is any error with neither subclip nor write_videofile. Is it possible to provide more details for the input video and also value for laser_on_frames? Thanks!

Task3:

This is the submitted artifacts for the task 3 mentioned above. It is trying to fix a bug from github a issue (https://github.com/Zulko/moviepy/issues/1816) The following is the link for pull requests: https://github.com/Zulko/moviepy/pull/1879

Here is a simple screenshot of the pull request.



Fix: with_duration check boundary for the parameter duration #1879

Open HikariDawn777 wants to merge 2 commits into `Zulko:master` from `HikariDawn777:fix_with_duration`

HikariDawn777 commented 33 seconds ago

- [Y] I have provided code that clearly demonstrates the bug and that only works correctly when applying this fix
- ☐ I have added suitable tests demonstrating a fixed bug or new/changed feature to the test suite in `tests/`
- [Y] I have properly documented new or changed features in the documentation or in the docstrings
- [Y] I have properly explained unusual or unexpected code in the comments around it

HikariDawn777 added 2 commits 4 minutes ago

```
fix: with duration boundary check    28c86db
fix: with duration boundary check    7af4943
```

Add more commits by pushing to the **fix_with_duration** branch on HikariDawn777/moviepy.

Reviewers
No reviews
Still in progress? Convert to draft

Assignees
No one assigned

Labels
None yet

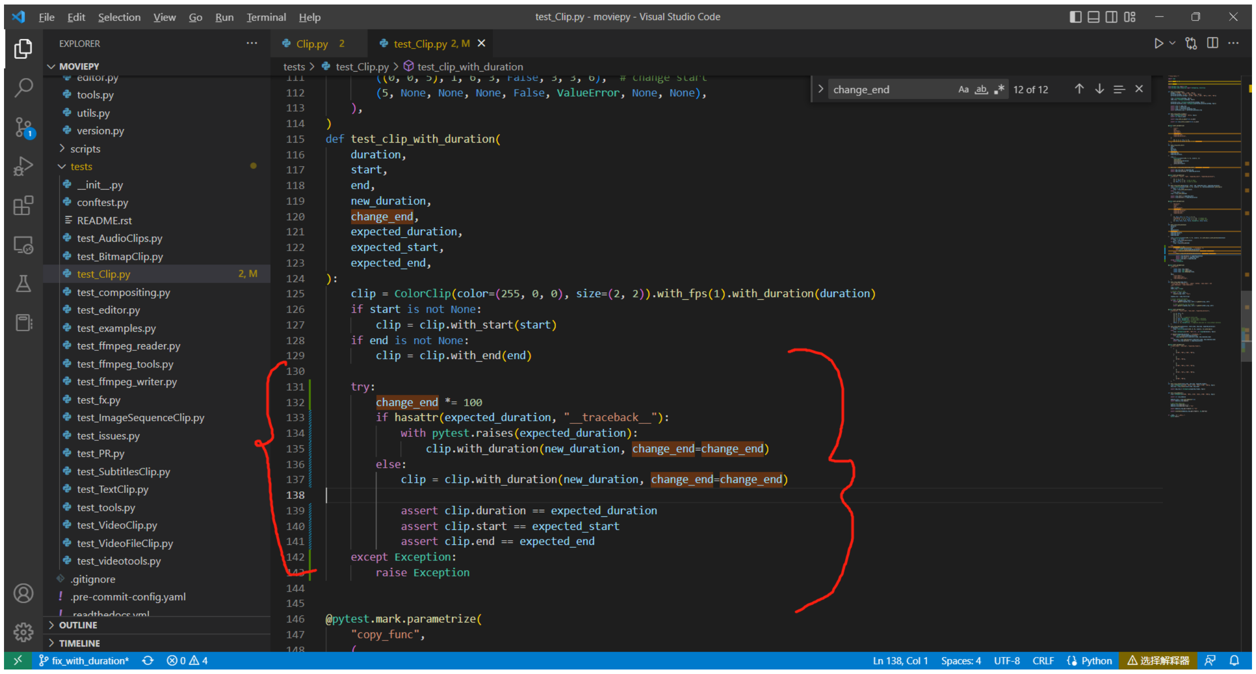
Projects
None yet

Milestone

Since the original bug issue in GitHub didn’t provide the exact video input that causes this problem, we think that this bug should be raised from the “duration” setting in MoviePy Clip.py function. And we did find that duration didn’t have the functionality to check boundaries and we decided to add such a patch.

The bug fix is to deal with the situation when a parameter named “duration” in the API “with_duration” is out of the range. It is trying to have a boundary check mechanism. If the “duration” parameter is larger than the video end time value (the largest possible), then it will raise a bug and stop the program. This is to some extent to protect the video editing not taking information out of the desired boundary.

We have written a test case (formal test case under the test folder) and it is the following screenshot. However, since we feel that the test case we wrote is not a very formal one (and may be hard to be accepted by maintainers), we just test it in our local environment. We didn't push and submit pull requests for this test case.



```python
        ((0, 0, 5), 1, 6, 3, False, 3, 3, 6),  # change start
        (5, None, None, None, False, ValueError, None, None),
    ),
)
def test_clip_with_duration(
    duration,
    start,
    end,
    new_duration,
    change_end,
    expected_duration,
    expected_start,
    expected_end,
):
    clip = ColorClip(color=(255, 0, 0), size=(2, 2)).with_fps(1).with_duration(duration)
    if start is not None:
        clip = clip.with_start(start)
    if end is not None:
        clip = clip.with_end(end)

    try:
        change_end *= 100
        if hasattr(expected_duration, "__traceback__"):
            with pytest.raises(expected_duration):
                clip.with_duration(new_duration, change_end=change_end)
        else:
            clip = clip.with_duration(new_duration, change_end=change_end)

            assert clip.duration == expected_duration
            assert clip.start == expected_start
            assert clip.end == expected_end
    except Exception:
        raise Exception


@pytest.mark.parametrize(
    "copy_func",
```

8. QA Strategy

We mainly design three different QA metrics to test our code. All of that QA strategy, including code style, unit testing, or multi-platform testing is performed for every task. We choose these two testing ways because they are important standards in the industry during our internship. Also, they are also used most frequently in academia for code checking when there is a new add-on feature or bug fix to existing code.

Code Style

As mentioned before, MoviePy uses certain style checks and guidelines called PEP8. PEP8 is a document that gives coding conventions for the Python code comprising the standard library in the main Python distribution. Before committing to our code, we ran PEP8 on our latest code to ensure our code adheres to this guideline. Also, the contributing guidelines of MoviePy, it also requires users to submit code under PEP8 guidelines, so this is why this testing is so essential for the contribution.

Unit testing & End-to-End Testing

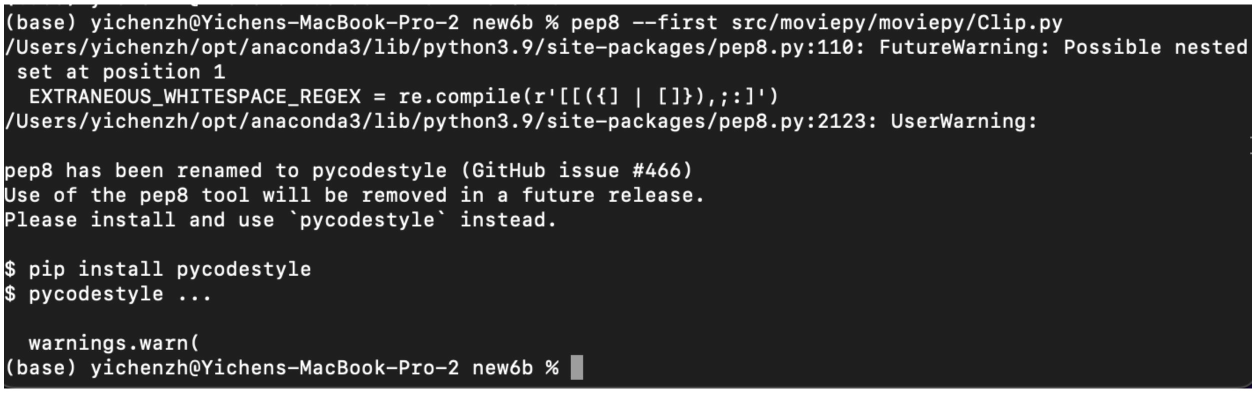
In standard MoviePy, there is a folder (named “tests”) that specifically lists a group of unit testing they need to test before each submission by typing “pytest” in the terminal. We think that this should be the place for unit testing and we tried to locate where the file that is in charge of our modification and try to imitate the modification like other methods under the same file. For the end-to-end testing, we have written a large python testing file to test MoviePy under real-world production cases that directly need to use such API and we find a real-world video input to test its performance and see if there are any bugs or if our code is activated. For both testing tools, if our code could pass all the test cases, we have strong confidence that there is nearly no bug in our implemented code.

9. QA Evidence

Code Style Result:

Code Style of Task 1

We performed the PEP8 code style test on task 1’s modify file, /src/moviepy/moviepy/Clip.py. From the command line output, we can see that our code has passed all the tests from PEP8 style checking tools (since it didn’t show any error from the screenshot, just showed some warnings about updating the PEP8 packages). So we have confidence that our task 1 has passed the PEP8 style check.



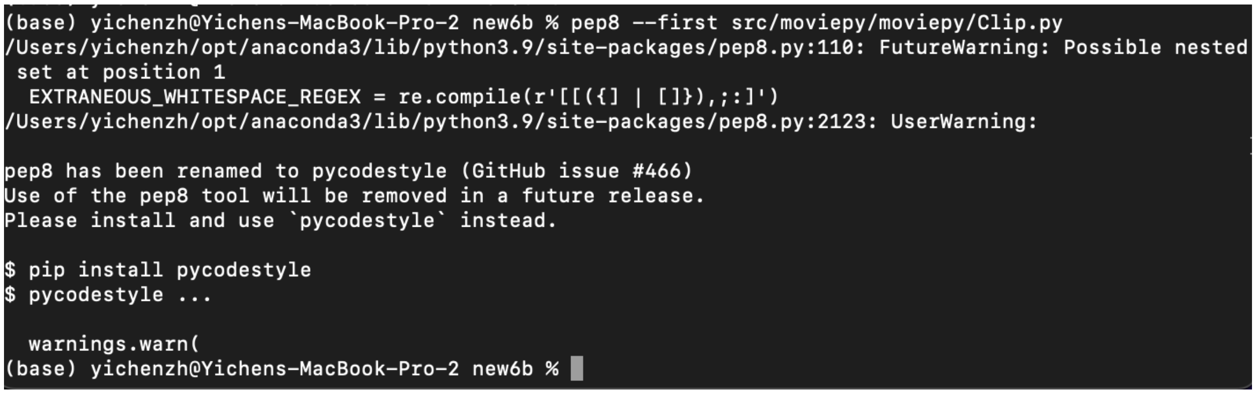
```
(base) yichenzh@Yichens-MacBook-Pro-2 new6b % pep8 --first src/moviepy/moviepy/Clip.py
/Users/yichenzh/opt/anaconda3/lib/python3.9/site-packages/pep8.py:110: FutureWarning: Possible nested
 set at position 1
  EXTRANEOUS_WHITESPACE_REGEX = re.compile(r'[[({] | []}),;:]')
/Users/yichenzh/opt/anaconda3/lib/python3.9/site-packages/pep8.py:2123: UserWarning:

pep8 has been renamed to pycodestyle (GitHub issue #466)
Use of the pep8 tool will be removed in a future release.
Please install and use `pycodestyle` instead.

$ pip install pycodestyle
$ pycodestyle ...

  warnings.warn(
(base) yichenzh@Yichens-MacBook-Pro-2 new6b %
```

Code Style of Task 2

As mentioned in the section ‘Submit Artifact’, we have removed Task 2 from our target list and chosen Task 3 instead. So there is no QA Evidence for Task 2.

Code Style of Task 3

Since both Task 1 and Task 3 need to modify the /src/moviepy/moviepy/Clip.py for either adding features or resolving bugs, so we only need to do the PEP8 checking once for both Task 1 and Task 3.

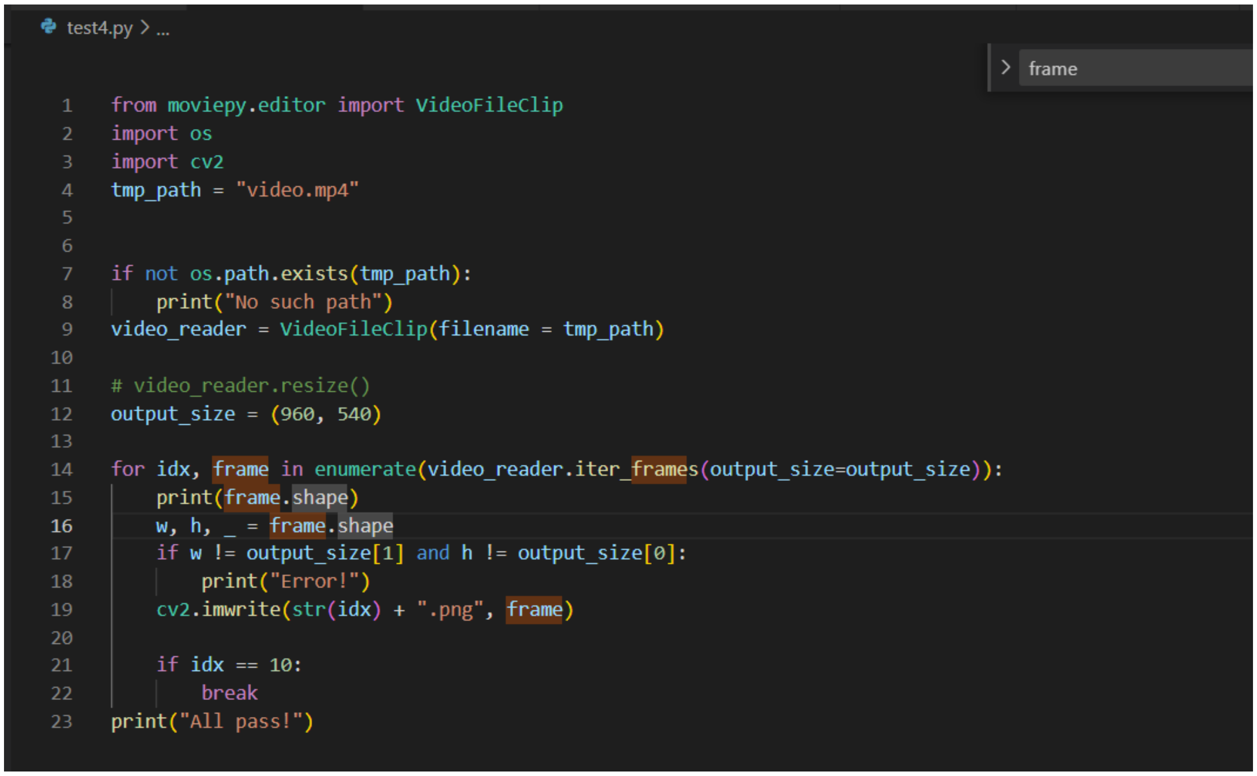


```
(base) yichenzh@Yichens-MacBook-Pro-2 new6b % pep8 --first src/moviepy/moviepy/Clip.py
/Users/yichenzh/opt/anaconda3/lib/python3.9/site-packages/pep8.py:110: FutureWarning: Possible nested
 set at position 1
  EXTRANEOUS_WHITESPACE_REGEX = re.compile(r'[[({] | []}),;:]')
/Users/yichenzh/opt/anaconda3/lib/python3.9/site-packages/pep8.py:2123: UserWarning:

pep8 has been renamed to pycodestyle (GitHub issue #466)
Use of the pep8 tool will be removed in a future release.
Please install and use `pycodestyle` instead.

$ pip install pycodestyle
$ pycodestyle ...

  warnings.warn(
(base) yichenzh@Yichens-MacBook-Pro-2 new6b %
```

Unit testing & End-to-End Testing Result:

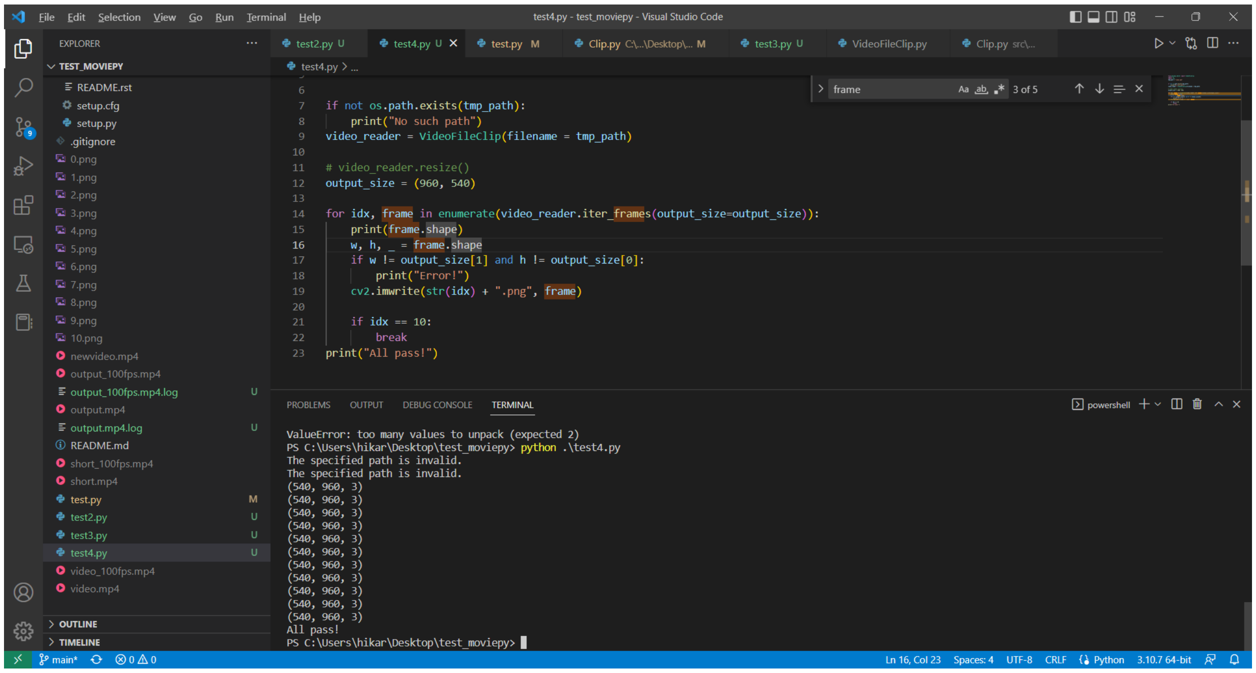
End-to-End Testing of Task 1:

We performed end-to-end testing for task 1 to verify our new feature's effectiveness. We chose the ten first frames of a random input video and tried modifying its output size with a preset ouput_size value. From the result, we can see that the output size after getting from iter_frames has the same size as our target, so the final performance is effective. Our test code is in Figure 9.2(a), and the test result is in Figure 9.2(b).



```python
from moviepy.editor import VideoFileClip
import os
import cv2
tmp_path = "video.mp4"


if not os.path.exists(tmp_path):
    print("No such path")
video_reader = VideoFileClip(filename = tmp_path)

# video_reader.resize()
output_size = (960, 540)

for idx, frame in enumerate(video_reader.iter_frames(output_size=output_size)):
    print(frame.shape)
    w, h, _ = frame.shape
    if w != output_size[1] and h != output_size[0]:
        print("Error!")
    cv2.imwrite(str(idx) + ".png", frame)

    if idx == 10:
        break
print("All pass!")
```

Figure 9.2(a)



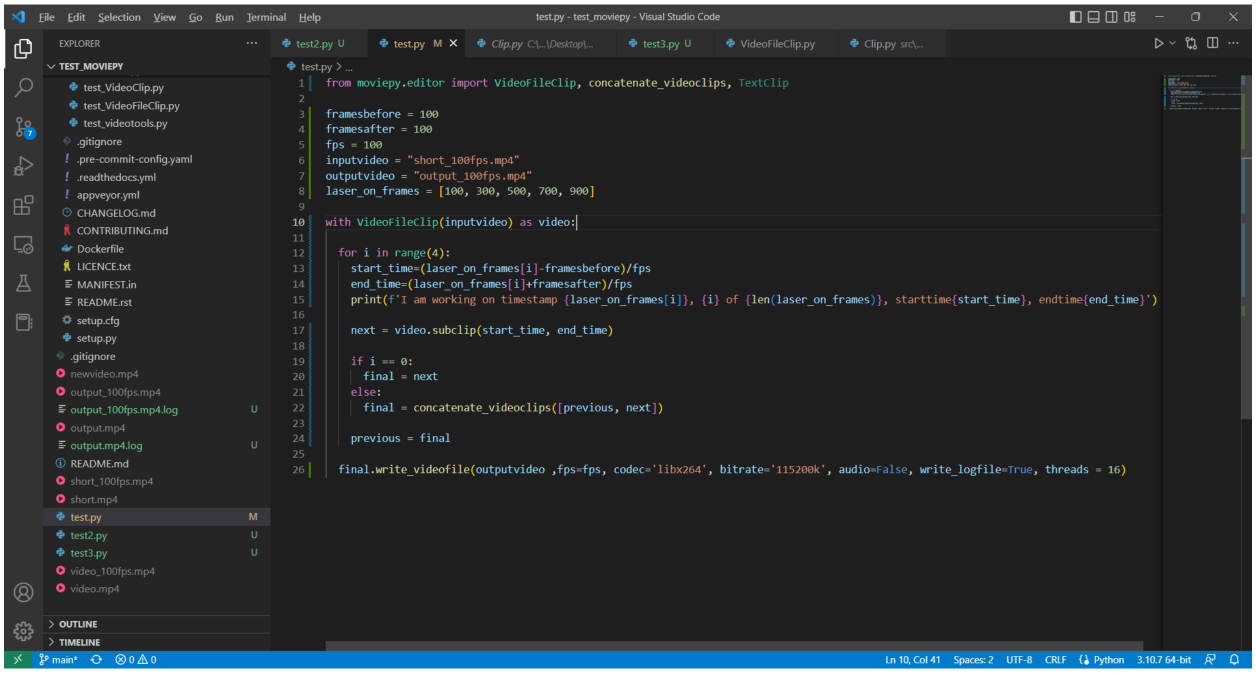
```python

if not os.path.exists(tmp_path):
    print("No such path")
video_reader = VideoFileClip(filename = tmp_path)

# video_reader.resize()
output_size = (960, 540)

for idx, frame in enumerate(video_reader.iter_frames(output_size=output_size)):
    print(frame.shape)
    w, h, _ = frame.shape
    if w != output_size[1] and h != output_size[0]:
        print("Error!")
    cv2.imwrite(str(idx) + ".png", frame)

    if idx == 10:
        break
print("All pass!")
```

```
ValueError: too many values to unpack (expected 2)
PS C:\Users\hikar\Desktop\test_moviepy> python .\test4.py
The specified path is invalid.
The specified path is invalid.
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
(540, 960, 3)
All pass!
PS C:\Users\hikar\Desktop\test_moviepy>
```

Figure 9.2(b)

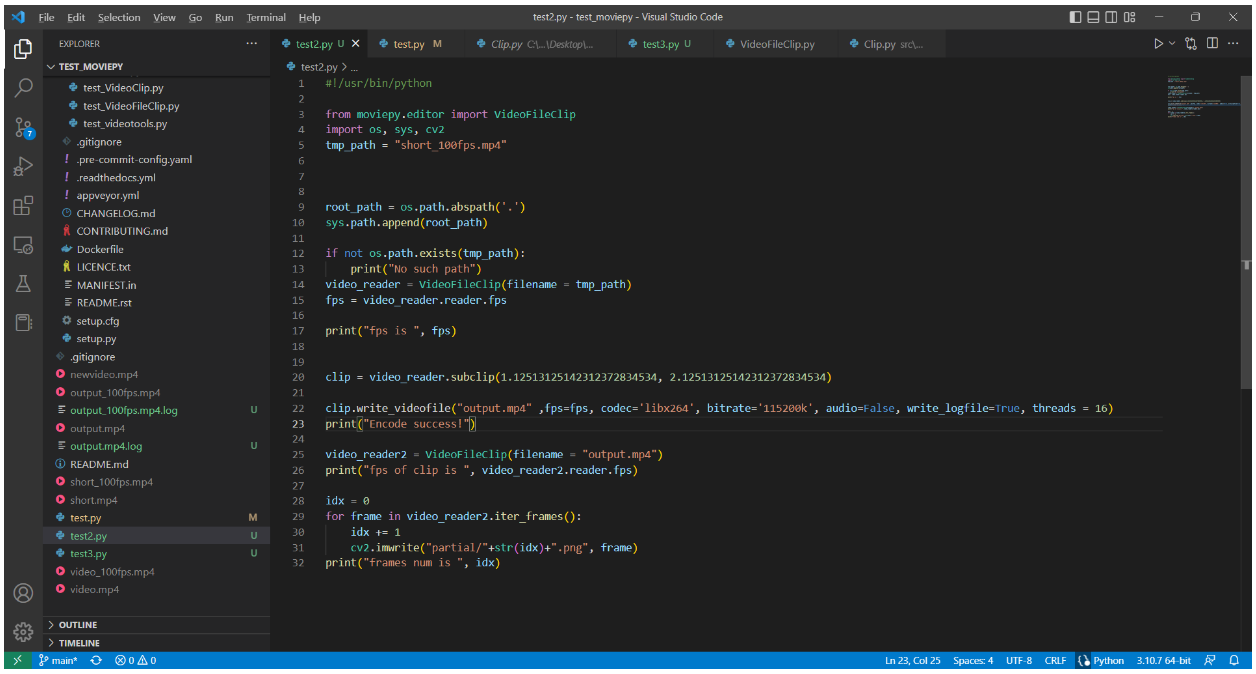
End-to-End Testing of Task 2:

The following two files are our **end-to-end** testing for task 2. We found an online real-world video (24 minutes) as its input source and see if all behavior lies in our expectations. We have used the following 2 test cases many times but we didn't find the error mentioned in the GitHub issues and here is our evidence for the end-to-end test cases we have written. (The test shown in Figure 9.3(a) is my personal test case and the Figure9.3(b) is trying to exactly use the code provided in that GitHub issues post)



```python
from moviepy.editor import VideoFileClip, concatenate_videoclips, TextClip

framesbefore = 100
framesafter = 100
fps = 100
inputvideo = "short_100fps.mp4"
outputvideo = "output_100fps.mp4"
laser_on_frames = [100, 300, 500, 700, 900]

with VideoFileClip(inputvideo) as video:

  for i in range(4):
    start_time=(laser_on_frames[i]-framesbefore)/fps
    end_time=(laser_on_frames[i]+framesafter)/fps
    print(f'I am working on timestamp {laser_on_frames[i]}, {i} of {len(laser_on_frames)}, starttime{start_time}, endtime{end_time}')

    next = video.subclip(start_time, end_time)

    if i == 0:
      final = next
    else:
      final = concatenate_videoclips([previous, next])

    previous = final

  final.write_videofile(outputvideo ,fps=fps, codec='libx264', bitrate='115200k', audio=False, write_logfile=True, threads = 16)
```

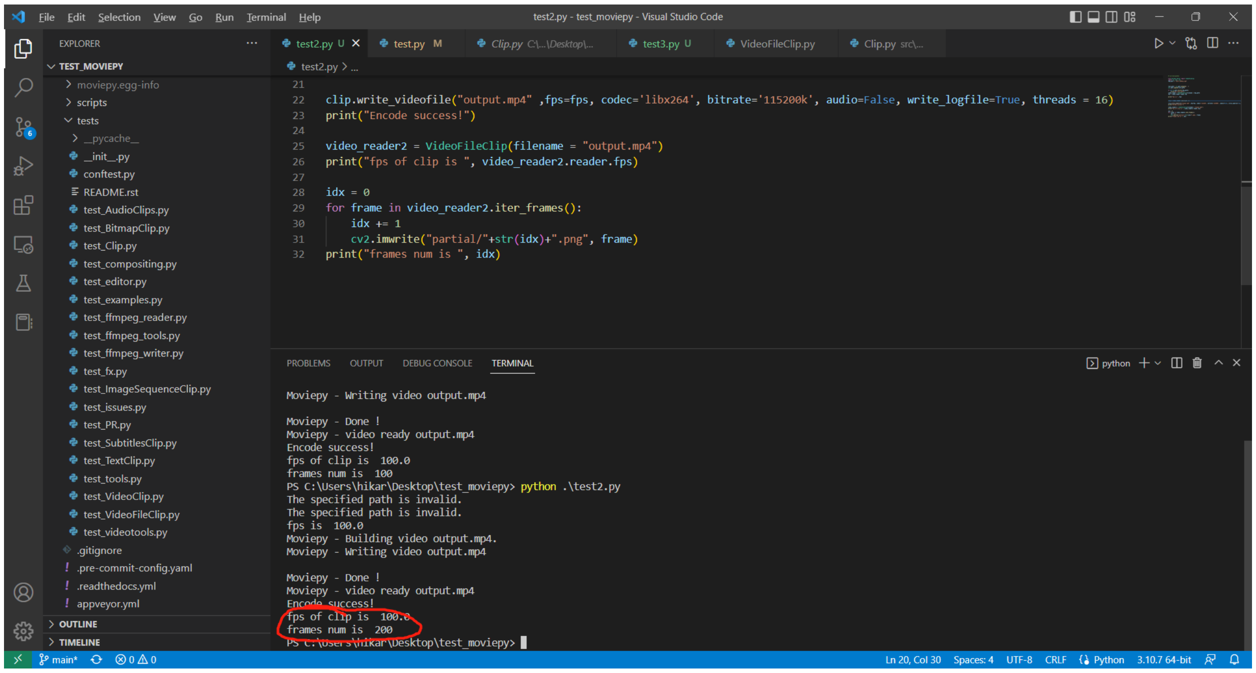
Figure 9.3(a)



```python
#!/usr/bin/python

from moviepy.editor import VideoFileClip
import os, sys, cv2
tmp_path = "short_100fps.mp4"



root_path = os.path.abspath('.')
sys.path.append(root_path)

if not os.path.exists(tmp_path):
    print("No such path")
video_reader = VideoFileClip(filename = tmp_path)
fps = video_reader.reader.fps

print("fps is ", fps)


clip = video_reader.subclip(1.12513125142312372834534, 2.12513125142312372834534)

clip.write_videofile("output.mp4" ,fps=fps, codec='libx264', bitrate='115200k', audio=False, write_logfile=True, threads = 16)
print("Encode success!")

video_reader2 = VideoFileClip(filename = "output.mp4")
print("fps of clip is ", video_reader2.reader.fps)

idx = 0
for frame in video_reader2.iter_frames():
    idx += 1
    cv2.imwrite("partial/"+str(idx)+".png", frame)
print("frames num is ", idx)
```

Figure 9.3(b)

The following figure 9.3(c) is the running result of these test cases and we can see from the red circle that the output frame number is actually 200 frames instead of the 210 frames mentioned in the GitHub issues. This is the evidence of why we finally think that such GitHub issues aren't a problem and reflect it back to the GitHub community.



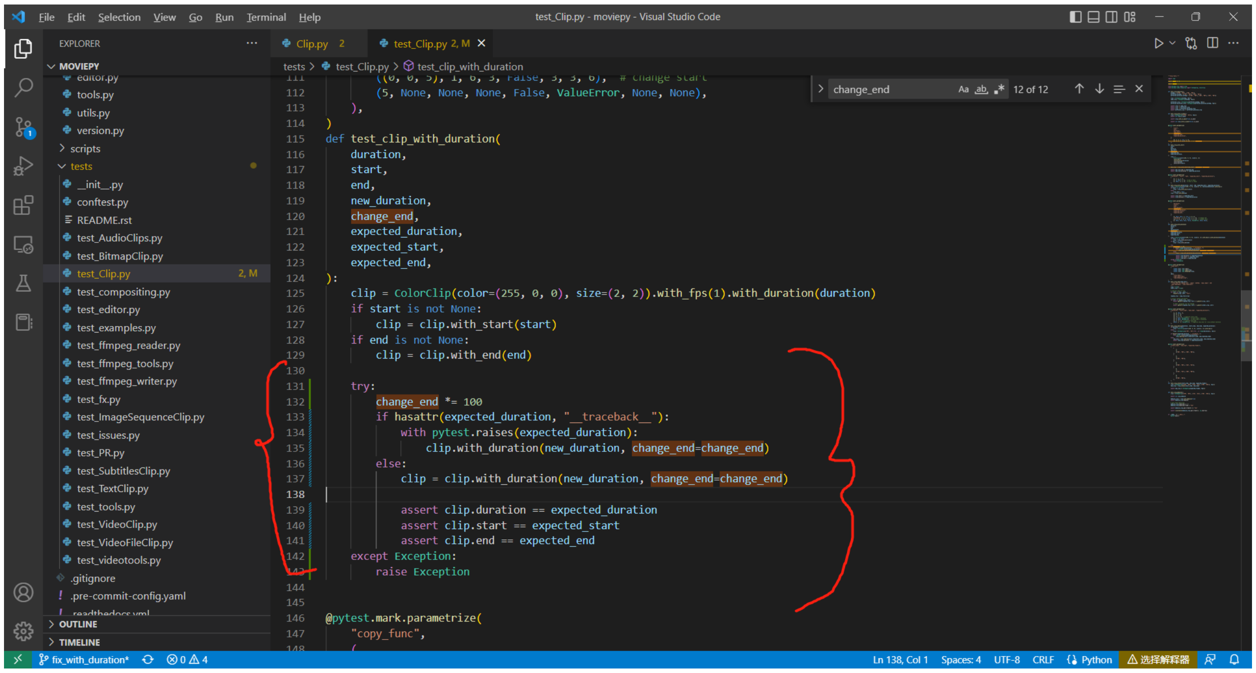
```python
clip.write_videofile("output.mp4" ,fps=fps, codec='libx264', bitrate='115200k', audio=False, write_logfile=True, threads = 16)
print("Encode success!")

video_reader2 = VideoFileClip(filename = "output.mp4")
print("fps of clip is ", video_reader2.reader.fps)

idx = 0
for frame in video_reader2.iter_frames():
    idx += 1
    cv2.imwrite("partial/"+str(idx)+".png", frame)
print("frames num is ", idx)
```

```
Moviepy - Writing video output.mp4

Moviepy - Done !
Moviepy - video ready output.mp4
Encode success!
fps of clip is  100.0
frames num is  100
PS C:\Users\hikar\Desktop\test_moviepy> python .\test2.py
The specified path is invalid.
The specified path is invalid.
fps is  100.0
Moviepy - Building video output.mp4.
Moviepy - Writing video output.mp4

Moviepy - Done !
Moviepy - video ready output.mp4
Encode success!
fps of clip is  100.0
frames num is  200
PS C:\Users\hikar\Desktop\test_moviepy>
```

Figure 9.3(c)

Unit testing of TASK3:

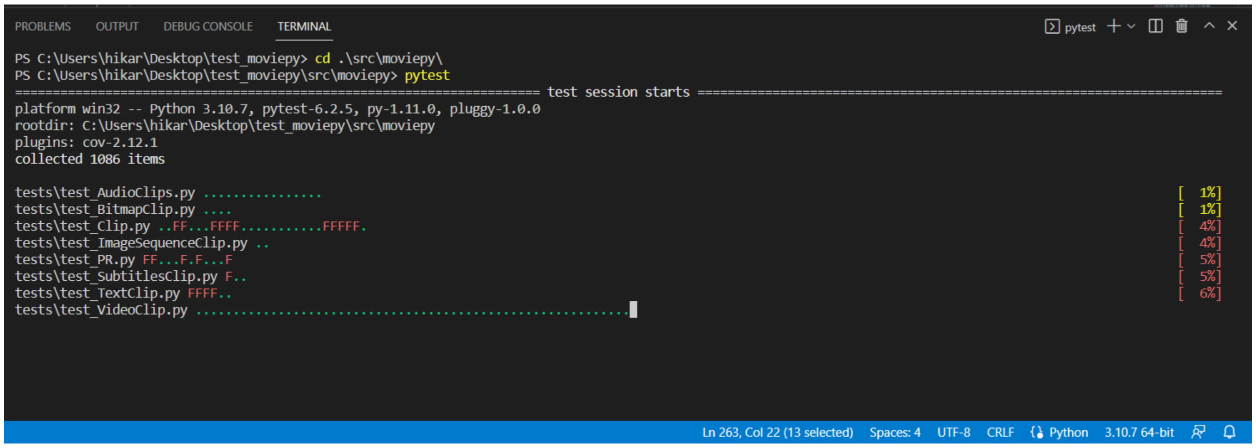
The following figure 9.4(a) is the **unit test** case we tried to modify. We found that “with_duration” API testing is under the test_Clip.py file and the method that is directly related to the “with_duration” API that we want to change is under “test_clip_with_duration”, so this is where we change. We multiplied the duration time to 100 times its original value because we want to raise the bug if it is out of range.



```python
        ((0, 0, 5), 1, 6, 3, False, 3, 3, 6),  # change start
        (5, None, None, None, False, ValueError, None, None),
    ),
)
def test_clip_with_duration(
    duration,
    start,
    end,
    new_duration,
    change_end,
    expected_duration,
    expected_start,
    expected_end,
):
    clip = ColorClip(color=(255, 0, 0), size=(2, 2)).with_fps(1).with_duration(duration)
    if start is not None:
        clip = clip.with_start(start)
    if end is not None:
        clip = clip.with_end(end)

    try:
        change_end *= 100
        if hasattr(expected_duration, "__traceback__"):
            with pytest.raises(expected_duration):
                clip.with_duration(new_duration, change_end=change_end)
        else:
            clip = clip.with_duration(new_duration, change_end=change_end)

            assert clip.duration == expected_duration
            assert clip.start == expected_start
            assert clip.end == expected_end
    except Exception:
        raise Exception


@pytest.mark.parametrize(
    "copy_func",
    (
```

Figure 9.4(a)

The following figure 9.4(b) is our unit testing evidence after running MoviePy's built-in testing system. Only need to care about the third line ("tests\test_clip.py") because this is the only file we edit and it seems that when there are some bugs, it successfully catches them.



```
PS C:\Users\hikar\Desktop\test_moviepy> cd .\src\moviepy\
PS C:\Users\hikar\Desktop\test_moviepy\src\moviepy> pytest
======================================================================= test session starts ========================================================================
platform win32 -- Python 3.10.7, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: C:\Users\hikar\Desktop\test_moviepy\src\moviepy
plugins: cov-2.12.1
collected 1086 items

tests\test_AudioClips.py ................                                                                                                                    [  1%]
tests\test_BitmapClip.py ....                                                                                                                                [  1%]
tests\test_Clip.py ..FF...FFFF...........FFFFF.                                                                                                              [  4%]
tests\test_ImageSequenceClip.py ..                                                                                                                           [  4%]
tests\test_PR.py FF...F.F...F                                                                                                                                [  5%]
tests\test_SubtitlesClip.py F..                                                                                                                              [  5%]
tests\test_TextClip.py FFFF..                                                                                                                                [  6%]
tests\test_VideoClip.py .........................................................
```

Figure 9.4(b)

10. Plan Updates

Specific Task Name	Task Description	Expected Hours	Actual Hours	Reason
Go over the codebase to localize task 1	We want to add a new feature that supports output video size in task 1, and we want to localize the area to add this feature.	2.5 hrs	8 hrs	We originally thought that we only needed to understand the code logic of MoviePy, because the code volume of MoviePy is not large. But after an in-depth investigation, we found that the class of FFMPEG is embedded in MoviePy, and there are many inheritance and function overrides, so we spent a lot of time studying FFMPEG and how this class interacts with MoviePy.
Fix the bug with Task 2	We try to solve the bug of Extra frames at the beginning and dropped frames at the end write_videofile, which is a GitHub issue raised by others. Our first step is to reproduce the bug, then	12 hrs	6 hrs	We have been unable to reproduce the bug shown in this GitHub issue. We tried on MacOS and windows respectively, and tried to contact the person who raised the bug, but did not

	analyze the cause of the error, and finally make a modification plan.			receive a reply, so we decided to give up solving this task2 and create a new task3.
Study and discover the knowledge for video processing tool	We should spend some time briefly learning the cs knowledge related to MoviePy, that is, the simple knowledge of video processing and analysis.	4 hrs	10 hrs	We found that to have an in-depth understanding of MoviePy, it is not enough to simply understand the meaning of some parameters of video production but to conduct in-depth research on video compression algorithms and FFMPEG packages. We spend at least twice as long on this.

11. Experiences and Recommendations

Overall Experience

Overall, this project is full of challenges but has brought us a lot of gains, and it has verified the theoretical knowledge learned in EECS 481 from practice. We were surprised to find that the top-down comprehension of code allows us to quickly get started with unfamiliar code, and the knowledge of design patterns also allows us to find the required knowledge more quickly from nested classes. The process of contributing open-source code is complex, and the problems to be solved are diverse. We found that effective communication with teammates can improve productivity, and we have also honed our software engineering capabilities from continuous problem-solving.

Previous experience

Before doing this whole GitHub repository, we found that the backend of the whole MoviePy is based on FFmpeg. This means that in the backend of MoviePy, it is continuously trying to call FFmpeg to achieve its functionality. Thus, to best understand and run the code of FFmpeg, we have prepared a lot of FFmpeg pre-knowledge, like how to cut part of a video clip directly by FFmpeg, how to change resolution and fps by FFmpeg, how to slip video to frames by FFmpeg, how to combine frames to video by FFmpeg, how to extract I frame from FFmpeg, and how to read video encode and decode information from FFmpeg. In order to do all those mentioned above, we have to check FFmpeg documents and check stack overflow to understand how to best exert such functionality without destroying important information of video input source (for example, if sometimes you set up incorrectly, you may set up a different codec, fps, or resolution and this will be a great hurt to test cases inputs).

Development Environment

In terms of computer configuration, one of us uses a windows computer configured with Intel Core i5, 16GB 2133 Mhz LPDDR3 RAM, and 256GB SSD and the other uses a 2017 Macbook Pro 13”, with 2.3 GHz dual-core Mac computer. The two of us Both use vscode for code development, and GitHub as the code management warehouse. In the development environment, we all use the Python virtual environment to avoid possible problems caused by package dependencies.

Communication with Maintenance

Unlike many open-source code contributors who use slack or discord for convenient communication, MoviePy contributors mainly use Github for communication. For longer or more in-depth discussions, developers use MoviePy Gitter. We first contacted the developer by e-mail and roughly described our motivation and solution for maintaining this project. The developer Zulko replied to the email a few days later, expressing his support for our idea, which made us very excited, and we also realized that our contribution will be recognized by others. When completing tasks, we mainly propose some ideas in the MoviePy Gitter, and then developers and well-wishers will occasionally make some suggestions for our improvements. In the end, our code was submitted through GitHub Pull Request, but because of the MoviePy contribution process, our code will be sent to multiple developers for review. This period of time may be relatively long, so we have not gotten notified that your code has been reviewed.

Trouble understanding, changing, and contributing

Actually, the whole process of contribution is hard for us to handle for the first time. Especially the pull requests, there may be multiple requirements and setups needed. However, since we have multiple tasks and we get the task easier and easier when we are doing more work. We feel pleased with the result. We believe that in the future, it will be much easier to contribute to open source projects in other GitHub and this experience will guide us a lot.

Meanwhile, when we are understanding the code, we will feel really appreciated if there is a system design level document to tell what kinds of design patterns it used and what groups of files are the most important file in understanding the structure of the whole repository. The lack of explanation sometimes is the reason why university students at first find it hard to participate in the open source project because there is really a large gap between what we did at school and what industry generally used.

Unanticipated challenges

There are actually tons of unanticipated problems coming out when we are writing the code. Sometimes, we think that our solution may be good, but when we test it on test cases, we find that it is wrong and we have to keep writing it until the test cases pass. It is easy for developers to **feel** that they can pass the requirements, but actually, it is their wrong feeling. Developers should use evidence to prove their judgment instead of feeling.

Also, for the pull requests, what the maintainers provided is vague and may be deprecated version. We strictly followed their steps of requirements, but it didn't run successfully based on the virtual environment they provided and we sucked them for a whole night to repeat their steps on and on. However, after we read other people’s steps for pull requests, we found that there may be some misunderstanding of their pull requests guideline and we chose an easier way to handle the push and merge and pull requests. We found that sometimes, we don’t necessarily need to strictly follow anything in the guideline, and instead reading more sample code or actions will be more helpful.

Team Collaboration and Useful feedback

Collaborative processes and culture among our team have helped a lot in our productivity. Through teamwork, my and my teammate have a unified project discussion every Sunday night. Following some productivity-enhancing laws, we limit our discussions to 3 hours. In the weekly meeting, my teammates and I first summarize the problems we solved and the difficulties we encountered this week, and then we discuss the difficulties we encountered together and try to solve a small part. Finally, we arrange the tasks we need to complete in the next week. In terms of project management tools, we mainly use slack to communicate, use notation to record important project documents, and use Gantt charts to manage project progress.

In terms of communication with developers, compared with other open-source projects, we think this project lacks some tools for developers to discuss freely, such as slack or discord. When asking questions in the in-depth discussion Gitter channel, we have had 2 questions that have not been answered, and the 2 questions that have been answered have not received a reply until about four or five days after the question was asked. In a certain sense, we think MoviePy's open-source contribution culture is not particularly helpful for us to improve efficiency. But we tried to make up for this deficiency with efficient cooperation among team members, and successfully completed the task on time.

What works

It is worth mentioning that MoviePy has done a good job in the standardization of the code, from the design pattern of the code, to the naming of variables, and to the standardization of comments and documents. During our research and development process, I also worked hard to learn the standardization of MoviePy in terms of code, and try to make the code we write as concise and standardized as possible.

What does not work and what we will do

It doesn't work well on library installation. In the past, what we did not work on most was environment setup issues. In the past, python libraries were controlled by the python environment itself. There are a lot of setup issues and we have to edit a lot of codes to adjust for it for the directory and dependencies issues. Moreover, we found that the unit testing provided by maintainers are little bit deprecated because it stops running somewhere in the middle when you put in "pytest", but luckily, the test case covered our files and we didn't care about those files that are out of the purpose of our works.

Also, for task 2, we have tried multiple video formats that may raise bugs in the GitHub issues description (like video with 100fps and video with fewer I-frames in the middle and tried to combine new video from image frames). We guess this may be due to I-frames or GOP (group of picture) settings in the video codec. Sometimes, When we want to extract a video clip by FFmpeg, there may be several frame adjustments during the process of cutting clips from videos (may cut more frames or less). Generally, their adjustment is directly related to the video encoded on a group of pictures. However, here, we have tested some videos, but we didn't find any error with either subclip or write_videofile. At last, we have to report such a problem to GitHub maintainers and search for further assistance.

Changes you would make if you were starting a new project

The next time we contribute to open-source code, we should have the obvious process in mind. When we find an issue of interest that needs to be resolved, we should reproduce the bug the first time, rather than try to solve it, because this bug may be solved "by chance" when the developer is doing other maintenance. In addition, we will try to study more open-source community standards, because some standards may have stringent requirements for code editing, such as other library dependencies that cannot be used in the implementation process because this will increase the dependencies required by the entire library's total number of items. The most important point is that in this open source project contribution, we should do the open source project we like, not something that is easier to increase our resume experience. In the next project, we will put our minds at ease, calm down to study, and make more meaningful things.

12. Advice for Future Students

Yichen: This EECS will cover a lot of reading and explanations of various concepts. If you want to succeed in this class, it is recommended that you read the weekly readings carefully to connect with what the teacher said in class. For those confusing concepts, try using flashcards on Quizlet to memorize them.

Boyang:

1. Try to study more for open source community standards because some may have very strict requirements for code editing like cannot use other library dependencies during the implementation because that will increase the total number of dependencies required for the whole library.
2. Do your open source for what you like, instead of what is easier to increase your resume experience. Sometimes open source may need more contributions from people who are truly interested in this library and can exert long-term efforts into this library.
3. Try to talk more with the contributor community and write some letters through email to increase friendship and connection to these communities.
4. For the advice of this course, students should be better to actually participate in an internship before taking this one because so many software skills named in the course need to have more close experience in the industry such that they can understand those concepts easier, like design pattern, agile development, debugging tools, etc.

Reference

https://web.eecs.umich.edu/~weimerw/481/hw6/karrot-kollin-shultz.pdf
https://zulko.github.io/moviepy/
https://github.com/Zulko/moviepy