

# SCAV: Practice 3

## FFmpeg

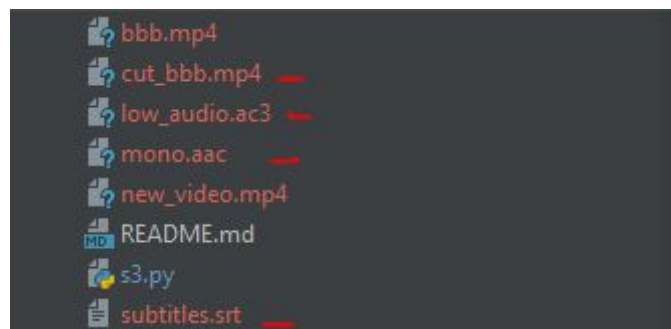
Iker Diaz Cilleruelo **194996**

---

### 1. You're going to create a new BBB container.

In order to create the different parts required, we have used some of the functions we already saw. To cut the video to 1 min, we have used the seeking operation. Then to get the audio in mono, we have first of all convert the video so it has only one channel like in the seminar 2, then, we have extracted the audio and obtained a .aac audio file. Similarly, with the audio in lower resolution, we have first extracted the 5.1 audio and then reduced the bitrate to 56k, then got a ac3 (as it is 5.1) audio file. Finally we have downloaded the subtitles from github in a srt format.

Then to pack everything we have used the code of the next section.



### 2. Create a python script able to automate the creation of MP4 containers. It can be like the previous one, more or less complex... be creative!

Following exercise one and what this one requires, we have done a dynamic code where the user can add as many streams as required as long as there is one video, one audio and one subtitle stream. Then we have used the -i ... command with the respective -c:v, -c:a and -c:s for video, audio and subtitles to copy into a new container.

```
Introduce streams to pack (input.mp4, audio.ac3, sub.srt, etc) and 0 to pack everything
cut_bbb.mp4
mono.aac
low_audio.ac3
subtitles.srt
0
```

```

Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'new_video.mp4':
Metadata:
  major_brand      : isom
  minor_version    : 512
  compatible_brands: isomiso2avc1mp41
  title           : Big Buck Bunny, Sunflower version
  artist          : Blender Foundation 2008, Janus Bager Kristensen 2013
  composer        : Sacha Goedegebure
  encoder         : Lavf58.45.100
  comment         : Creative Commons Attribution 3.0 - http://bbb3d.renderfarming.net
  genre           : Animation
Duration: 00:01:00.00, start: 0.000000, bitrate: 3406 kb/s
Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 1920x1080 [SAR 1:1 DAR 16:9], 3081 kb/s, 30 fps, 30 tbr, 30k tbn, 60 tbc (default)
Metadata:
  handler_name     : GPAC ISO Video Handler
Stream #0:1(und): Audio: ac3 (ac-3 / 0x332D6361), 48000 Hz, 5.1(side), fltp, 320 kb/s (default)
Metadata:
  handler_name     : GPAC ISO Audio Handler
Side data:
  audio service type: main
Stream #0:2(und): Subtitle: mov_text (tx3g / 0x67337874), 0 kb/s (default)
Metadata:
  handler_name     : SubtitleHandler
At least one output file must be specified

```

Probably there is something wrong with the subtitle file we have used, as when added to the container it has 0 bits.

### 3. Create a script which reads the tracks from an MP4 container, and it's able to say:

- Which broadcasting standard would fit
- ERROR in case it doesn't fit any
- Any more “pijada” you could think (be creative!)

To obtain the tracks from a container, we have used a function to read the `ffmpeg -i video.mp4` metadata fully. We search then for the streams on it. Rightnow we have only searched for the 3 first ones (as in the BBB video). Depending on the video, this would change, then as well, the subtitles must be controlled, as that stream is not relevant for the broadcasting standard, and could lead to errors if read. The process is the same as in the last practice, we look for the stream value on the .txt file with a `str.find`, then read the first value, which will be for instance, h264, aac, mp3, ac3, av1, etc. Then we store these tracks in the class tracks attribute and call the function with those codecs to find a suitable broadcasting format. The format should allow every codec. For the BBB video, we have:

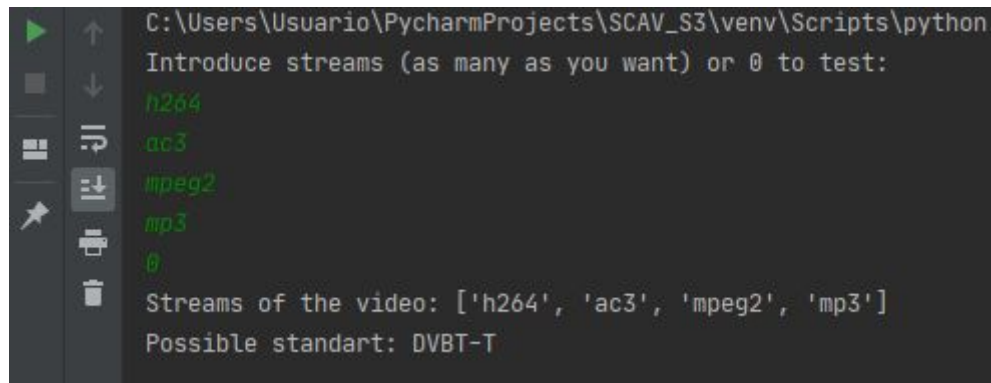
```

C:\Users\Usuario\PycharmProjects\SCAV_S3\venv\Scripts\python
Streams of the video: ['h264', 'mp3', 'ac3']
Possible standart: DVBT-T
Process finished with exit code 0

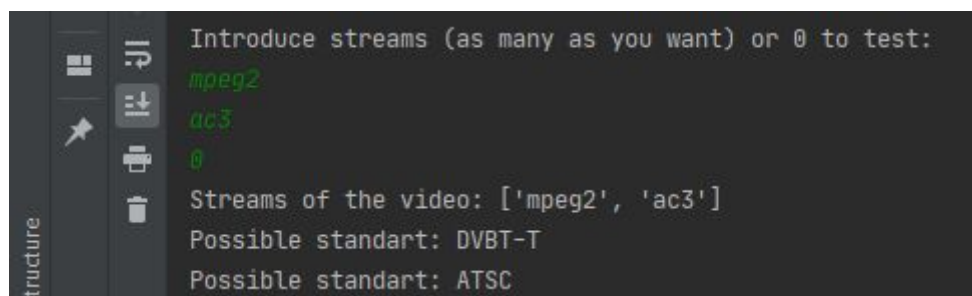
```

**4. Create a testing script, which will generate containers to launch against exercise '3'. You can reuse anything from previous lessons.**

To launch containers against the last exercise, the user can add as many streams as desired and then check the possible standard, pretty easy. It uses the same function, but this time the streams are not read from the metadata file of the video, but introduced by the user, and then compared with the availability in the different standards.



```
C:\Users\Usuario\PycharmProjects\SCAV_S3\venv\Scripts\python.  
Introduce streams (as many as you want) or 0 to test:  
h264  
ac3  
mpeg2  
mp3  
0  
Streams of the video: ['h264', 'ac3', 'mpeg2', 'mp3']  
Possible standart: DVBT-T
```



```
Introduce streams (as many as you want) or 0 to test:  
mpeg2  
ac3  
0  
Streams of the video: ['mpeg2', 'ac3']  
Possible standart: DVBT-T  
Possible standart: ATSC
```

**5. Integrate everything inside a class.**

Finally, the class integration. We are not going to comment too much about it. The objective was to have all in a class to showcase that we know how to use them. In our case we have created one attribute that is useful, the filename of the video, that will be used directly for the first operation, to read metadata, and then indirectly for the second and third operation. Then we have also created a tracks stream attribute to store the streams and use it in the third exercise. The class is not really useful, we mix some things and functions that are independent, despite that, we can use everything as expected.