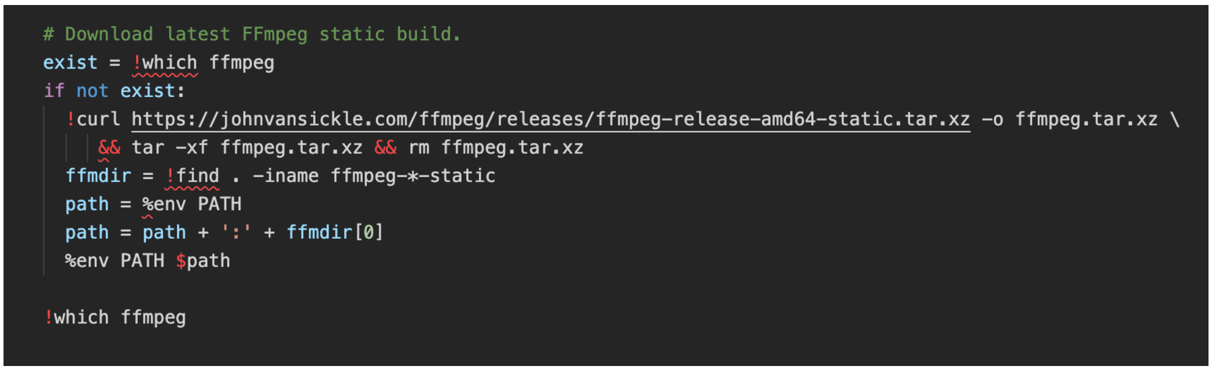
Exercise 3

The application can be found here, the file named **Exercise 3.ipynb**:

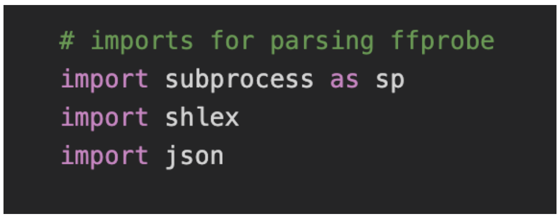
https://hub.labs.coursera.org:443/connect/sharedojwzxqru?forceRefresh=false

The ffmpeg and ffprobe installation and configuration happens in the first cell of the notebook. We specify if not already installed, to download a certain release of ffmpeg and to add it to our environments path at *usr//bin/ffmpeg*. This will allow us to use the two to check our files and adjust them, all with our base kernel.



```
# Download latest FFmpeg static build.
exist = !which ffmpeg
if not exist:
  !curl https://johnvansickle.com/ffmpeg/releases/ffmpeg-release-amd64-static.tar.xz -o ffmpeg.tar.xz \
     && tar -xf ffmpeg.tar.xz && rm ffmpeg.tar.xz
  ffmdir = !find . -iname ffmpeg-*-static
  path = %env PATH
  path = path + ':' + ffmdir[0]
  %env PATH $path

!which ffmpeg
```

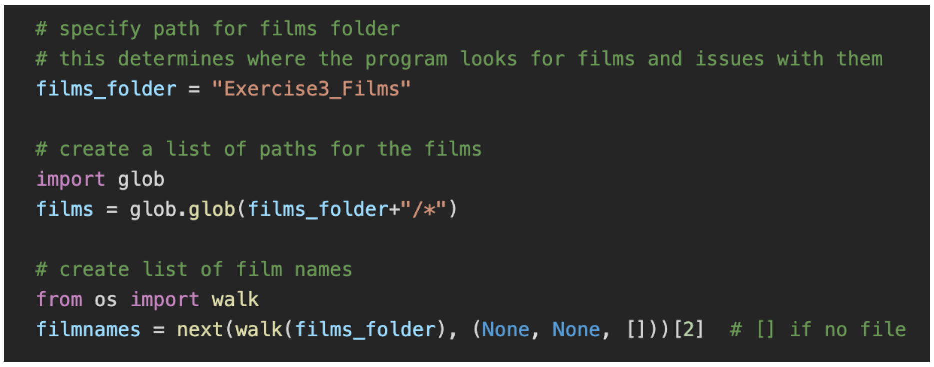
We handle the further imports, that I've specifically chosen as my solution for parsing ffprobe output and building subprocess ffmpeg statements for each file.



```
# imports for parsing ffprobe
import subprocess as sp
import shlex
import json
```

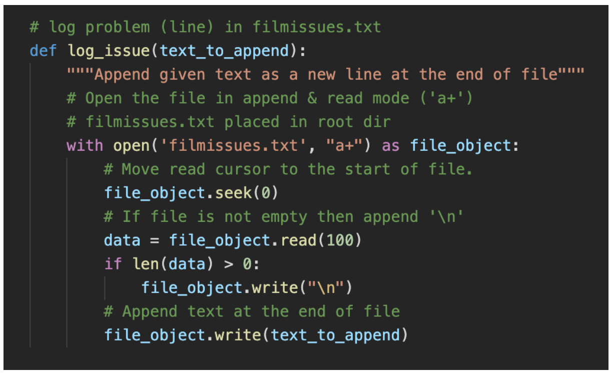
The application, as requested, automates the process of checking if a directory of submitted adhere to the festival's format and modify them if necessary. It does this in notebook format, so you'll have to run the cells successively and ensure the file path for the films matches your directory. I have uploaded the Exercise3_Films folder in case you do not want to change anything.

The application takes a film folder, and for ease of use for the functions later, creates a list of both the file paths of the films and their names. This is achieved with python's glob and os.walk, which iterate over the folder making their own lists.

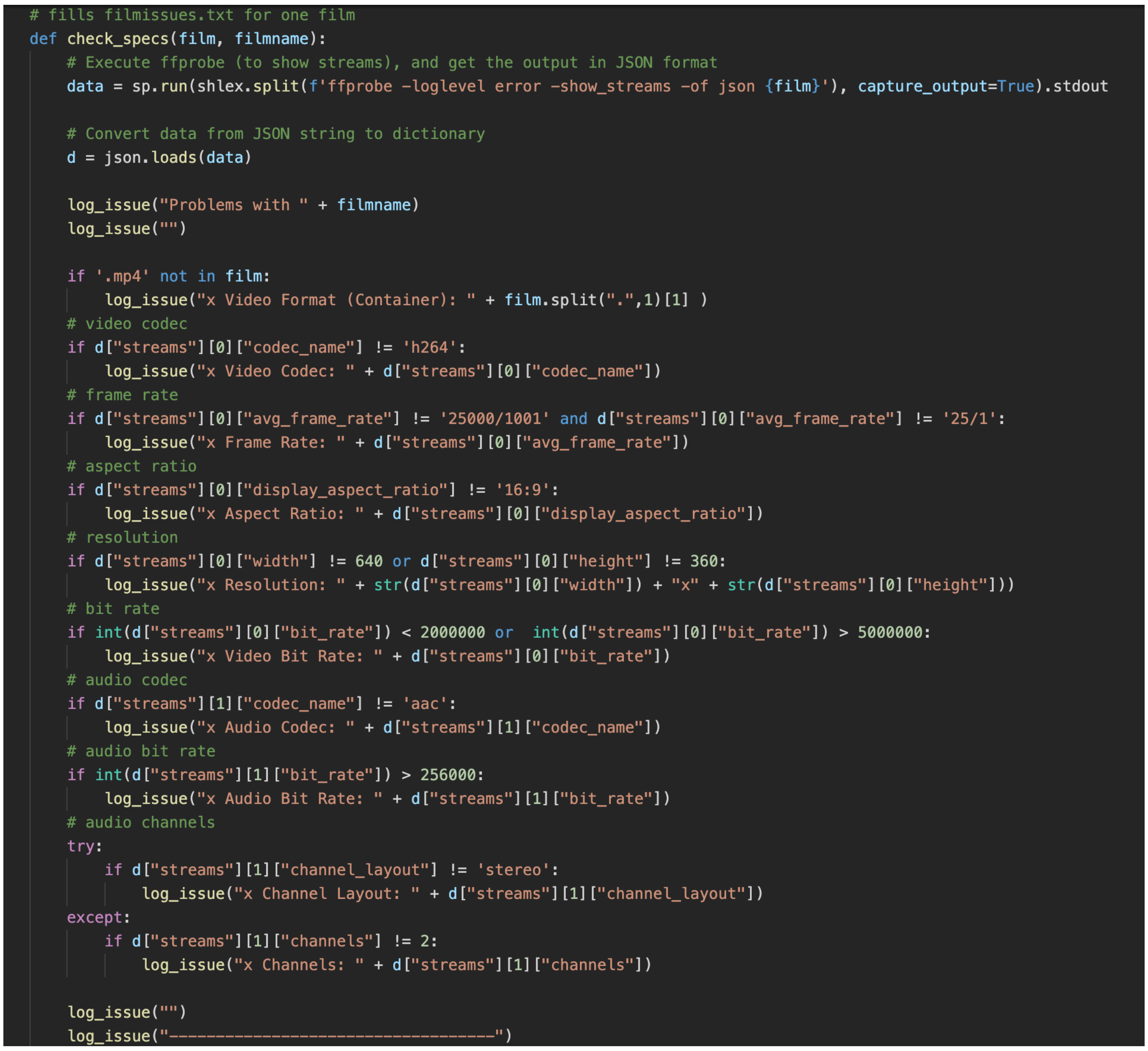


```python
# specify path for films folder
# this determines where the program looks for films and issues with them
films_folder = "Exercise3_Films"

# create a list of paths for the films
import glob
films = glob.glob(films_folder+"/*")

# create list of film names
from os import walk
filmnames = next(walk(films_folder), (None, None, []))[2]  # [] if no file
```

The log_issue and check_specs function take care of creating and writing to the filmissues.txt file, containing all differences between the submitted films and the specifications. It stores this txt file in the root directory (same as the notebook) of the application.



```python
# log problem (line) in filmissues.txt
def log_issue(text_to_append):
    """Append given text as a new line at the end of file"""
    # Open the file in append & read mode ('a+')
    # filmissues.txt placed in root dir
    with open('filmissues.txt', "a+") as file_object:
        # Move read cursor to the start of file.
        file_object.seek(0)
        # If file is not empty then append '\n'
        data = file_object.read(100)
        if len(data) > 0:
            file_object.write("\n")
        # Append text at the end of file
        file_object.write(text_to_append)
```

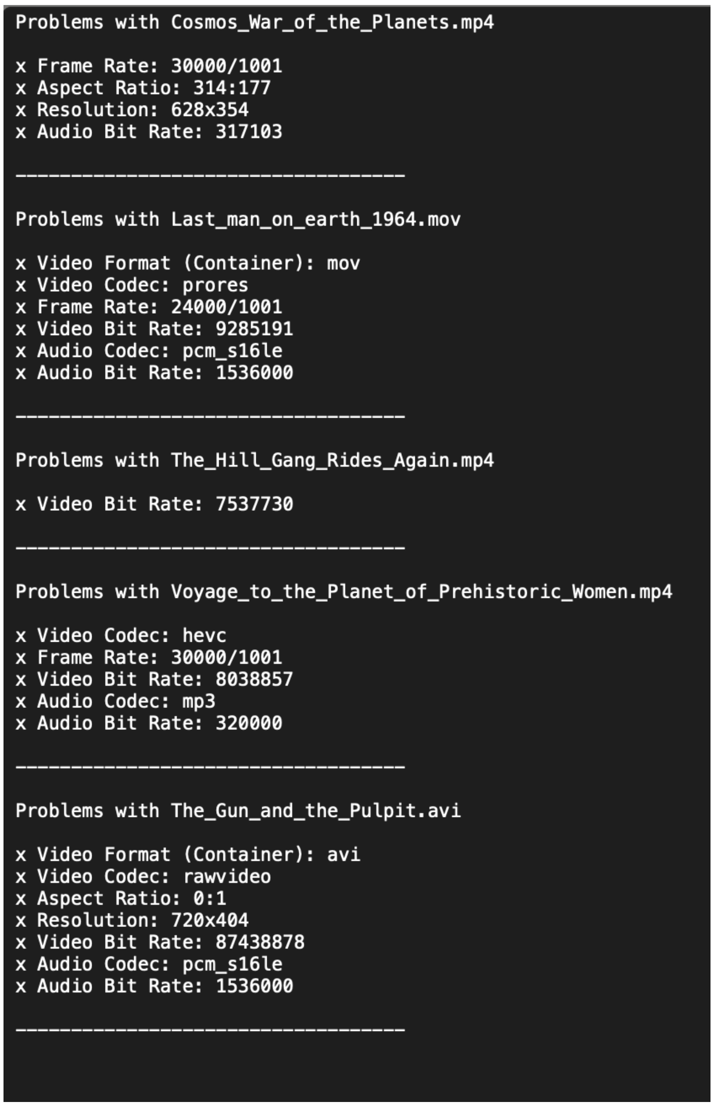


```python
# fills filmissues.txt for one film
def check_specs(film, filmname):
    # Execute ffprobe (to show streams), and get the output in JSON format
    data = sp.run(shlex.split(f'ffprobe -loglevel error -show_streams -of json {film}'), capture_output=True).stdout

    # Convert data from JSON string to dictionary
    d = json.loads(data)

    log_issue("Problems with " + filmname)
    log_issue("")

    if '.mp4' not in film:
        log_issue("x Video Format (Container): " + film.split(".",1)[1] )
    # video codec
    if d["streams"][0]["codec_name"] != 'h264':
        log_issue("x Video Codec: " + d["streams"][0]["codec_name"])
    # frame rate
    if d["streams"][0]["avg_frame_rate"] != '25000/1001' and d["streams"][0]["avg_frame_rate"] != '25/1':
        log_issue("x Frame Rate: " + d["streams"][0]["avg_frame_rate"])
    # aspect ratio
    if d["streams"][0]["display_aspect_ratio"] != '16:9':
        log_issue("x Aspect Ratio: " + d["streams"][0]["display_aspect_ratio"])
    # resolution
    if d["streams"][0]["width"] != 640 or d["streams"][0]["height"] != 360:
        log_issue("x Resolution: " + str(d["streams"][0]["width"]) + "x" + str(d["streams"][0]["height"]))
    # bit rate
    if int(d["streams"][0]["bit_rate"]) < 2000000 or  int(d["streams"][0]["bit_rate"]) > 5000000:
        log_issue("x Video Bit Rate: " + d["streams"][0]["bit_rate"])
    # audio codec
    if d["streams"][1]["codec_name"] != 'aac':
        log_issue("x Audio Codec: " + d["streams"][1]["codec_name"])
    # audio bit rate
    if int(d["streams"][1]["bit_rate"]) > 256000:
        log_issue("x Audio Bit Rate: " + d["streams"][1]["bit_rate"])
    # audio channels
    try:
        if d["streams"][1]["channel_layout"] != 'stereo':
            log_issue("x Channel Layout: " + d["streams"][1]["channel_layout"])
    except:
        if d["streams"][1]["channels"] != 2:
            log_issue("x Channels: " + d["streams"][1]["channels"])

    log_issue("")
    log_issue("-----------------------------------")
```

The checkspecs function uses shlex.split and a series of if statements to parse an ffprobe command for each file. It does this to check the following:

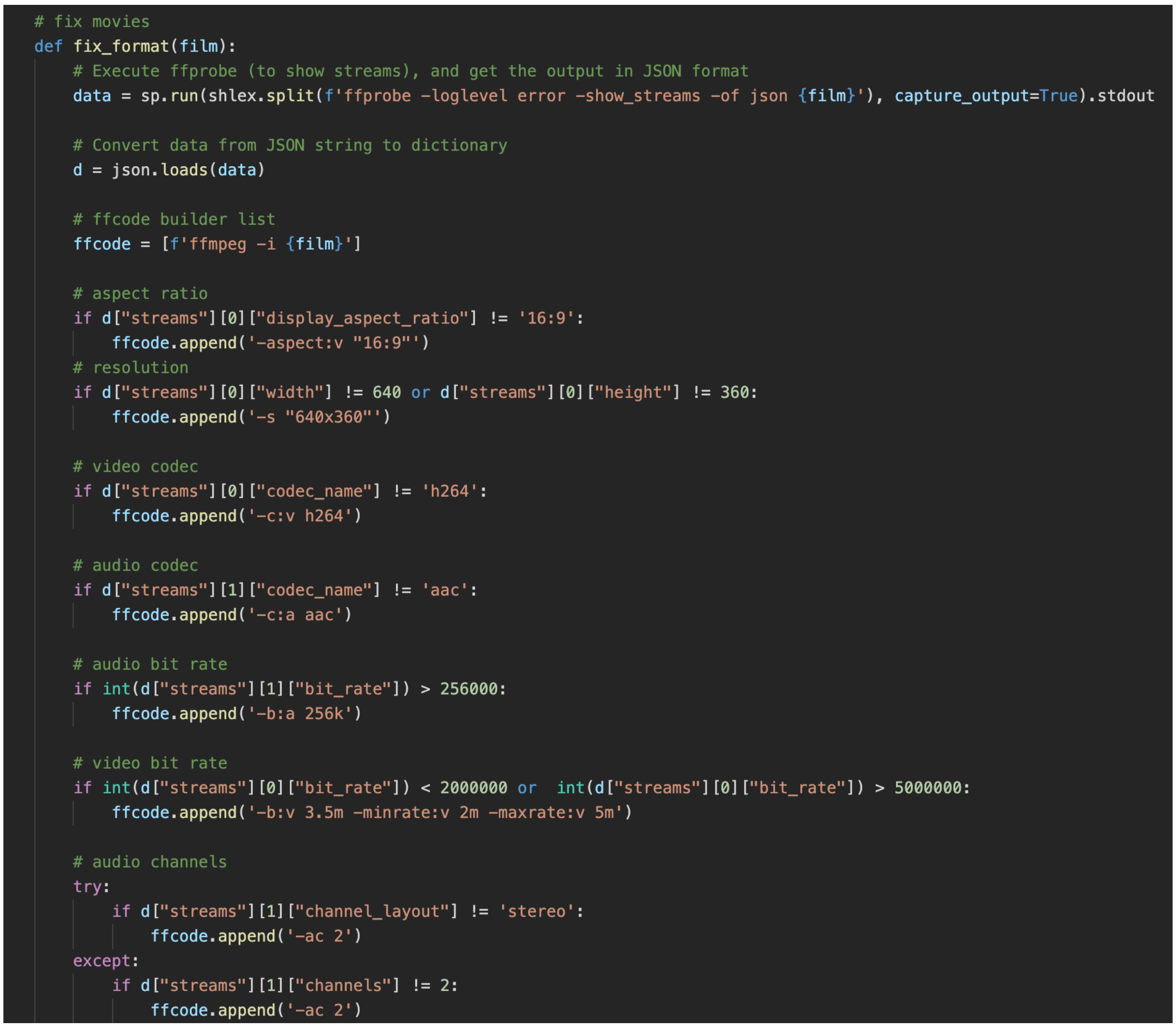
- **Video format (container)**: like a box that contains the video data and metadata, also most noticeable as the file’s extension (.mp4, .mov, etc)
- **Video codec**: video codecs are a software (group of algorithms) that compress and uncompress your video file
- **Audio codec**: similarly, audio codecs compress and uncompress audio files
- **Frame rate**: or frames per second, defines how many images are captured per second of a video file
- **Aspect Ratio**: the ratio between a videos height and width
- **Resolution**: one number, usually represented as two, to define the number of pixels in a video; for instance our specified resolution of 640x360 would have a fair 230400 pixels per image
- **Video bitrate**: the number of bits used to store a second of an video file; it generally determines the quality of the video, higher the bitrate, the better the quality
- **Audio bitrate**: similarly, the number of bits used to store a second of an audio file
- **Audio channels**: the number of channels in an audio file (stereo = 2, mono = 1)

The resulting filmissues.txt file looks like this.



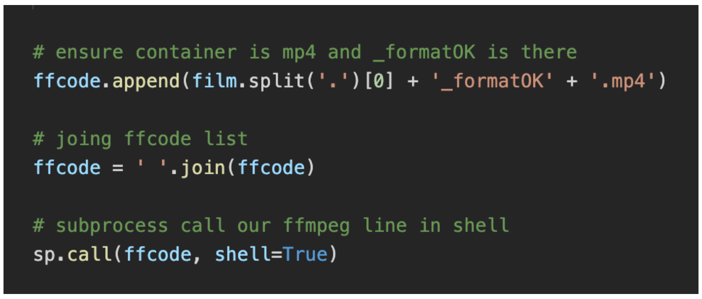
```
Problems with Cosmos_War_of_the_Planets.mp4

x Frame Rate: 30000/1001
x Aspect Ratio: 314:177
x Resolution: 628x354
x Audio Bit Rate: 317103

-----------------------------------

Problems with Last_man_on_earth_1964.mov

x Video Format (Container): mov
x Video Codec: prores
x Frame Rate: 24000/1001
x Video Bit Rate: 9285191
x Audio Codec: pcm_s16le
x Audio Bit Rate: 1536000

-----------------------------------

Problems with The_Hill_Gang_Rides_Again.mp4

x Video Bit Rate: 7537730

-----------------------------------

Problems with Voyage_to_the_Planet_of_Prehistoric_Women.mp4

x Video Codec: hevc
x Frame Rate: 30000/1001
x Video Bit Rate: 8038857
x Audio Codec: mp3
x Audio Bit Rate: 320000

-----------------------------------

Problems with The_Gun_and_the_Pulpit.avi

x Video Format (Container): avi
x Video Codec: rawvideo
x Aspect Ratio: 0:1
x Resolution: 720x404
x Video Bit Rate: 87438878
x Audio Codec: pcm_s16le
x Audio Bit Rate: 1536000

-----------------------------------
```

The fix_format function takes a film and probes it, much like the previous function, though instead, it builds an ffmpeg subprocess. It checks if the film fits the specifications, and where it does not, the function appends the necessary ffmpeg command.



```python
# fix movies
def fix_format(film):
    # Execute ffprobe (to show streams), and get the output in JSON format
    data = sp.run(shlex.split(f'ffprobe -loglevel error -show_streams -of json {film}'), capture_output=True).stdout

    # Convert data from JSON string to dictionary
    d = json.loads(data)

    # ffcode builder list
    ffcode = [f'ffmpeg -i {film}']

    # aspect ratio
    if d["streams"][0]["display_aspect_ratio"] != '16:9':
        ffcode.append('-aspect:v "16:9"')
    # resolution
    if d["streams"][0]["width"] != 640 or d["streams"][0]["height"] != 360:
        ffcode.append('-s "640x360"')

    # video codec
    if d["streams"][0]["codec_name"] != 'h264':
        ffcode.append('-c:v h264')

    # audio codec
    if d["streams"][1]["codec_name"] != 'aac':
        ffcode.append('-c:a aac')

    # audio bit rate
    if int(d["streams"][1]["bit_rate"]) > 256000:
        ffcode.append('-b:a 256k')

    # video bit rate
    if int(d["streams"][0]["bit_rate"]) < 2000000 or  int(d["streams"][0]["bit_rate"]) > 5000000:
        ffcode.append('-b:v 3.5m -minrate:v 2m -maxrate:v 5m')

    # audio channels
    try:
        if d["streams"][1]["channel_layout"] != 'stereo':
            ffcode.append('-ac 2')
    except:
        if d["streams"][1]["channels"] != 2:
            ffcode.append('-ac 2')
```

Later, the function ensures the name and container fit the specifications, joins the list we created, and runs the ffmpeg command as a subprocess within the shell.



```python
# ensure container is mp4 and _formatOK is there
ffcode.append(film.split('.')[0] + '_formatOK' + '.mp4')

# joing ffcode list
ffcode = ' '.join(ffcode)

# subprocess call our ffmpeg line in shell
sp.call(ffcode, shell=True)
```

Two for loops, one after check_specs and the other after fix_format, ensure this whole process happens for all films. The fixed films are put in the same directory as the original films, though with new names.

I designed the application as two separate sets of functions to isolate their concerns, and for easier grading. To automate this process further, I could design a main function that handles everything.