You need to create a Python library that can handle different folders such as birthday, anniversary, love, valentine, and wedding wishes. Each of these folders contains different types of pictures. The library should automate the process of generating videos, as it is designed to be asynchronous.

Here's how the library should work: it randomly selects one of the folders, then selects ten pictures from that folder randomly, and creates a video with those pictures. It also adds a song to the video. For the song, there are different folders available just like for images. The library should select a song from the same topic as the selected images.

Now, let's talk about effects and transitions. You'll need to create two classes in the library: one for effects and one for transitions. The effects class will have methods for different types of effects, and the library will randomly select one of these methods to add an effect to the video. The same process applies to transitions.

Some considerations: the duration of the video should match the duration of the song, and if the song is longer than one minute, it should be trimmed, and if it's shorter, it should be looped. Once the video is complete, The final video needs to be saved in the 'Generated video' folder, ensuring that it is saved in a folder with the same name as the one from which the images were taken when the video creation process started.

Another aspect to consider is the automation of uploading the generated videos to YouTube using Selenium. However, the logic for selecting the appropriate title, description, and tags based on the type of video needs to be implemented.

This is the basic outline of the script, and you'll need to fill in the details to meet all the requirements.

Grimoire

To create this Python library, we'll break down the task into smaller components and outline them in pseudocode before writing the actual code. This will help us ensure all functionalities are covered and the code is organized.

Pseudocode:

```
VideoCreator Library:
    Import necessary libraries (os, random, moviepy, etc.)
    Define paths for image and song folders.
    Create a VideoCreator class that handles video creation.
Folder Selection:
    Within VideoCreator, write a method to randomly select a folder.
Image Selection:
    In VideoCreator, write a method to randomly select ten images from the chosen folder.
Song Selection:
    Similarly, write a method to select a song from a folder matching the theme of the images.
Effects and Transitions Classes:
    Define two classes, Effects and Transitions, each containing multiple methods for different effects and transitions.
    In VideoCreator, write methods to randomly apply these effects and transitions between images.
Video Compilation:
    Write a method to create a video from the selected images, apply effects and transitions, and add the selected song as background music.
Video Finalization:
    Ensure the video duration matches the song duration, with trimming or looping as necessary.
    Save the final video in the appropriate 'Generated video' folder.
YouTube Upload:
    Use Selenium to automate video uploads to YouTube, with appropriate titles, descriptions, and tags based on the video type.
```

Now, let's start translating this into actual code.

Code:

First, we'll need to install the required libraries if you don't have them already:

```bash
pip install moviepy os-sys Pillow selenium
```

Now, the Python code:

```python
import os import random from moviepy.editor import ImageSequenceClip, AudioFileClip from
selenium import webdriver from selenium.webdriver.common.keys import Keys import time
class Effects: # Placeholder for effects methods def black_and_white(self, clip): return
clip.fx(vfx.blackwhite) # Add more effects here class Transitions: # Placeholder for
transition methods def fade_in(self, clip, duration=1): return clip.crossfadein(duration)
# Add more effects here class VideoCreator: def __init__(self): self.base_path =
'/path/to/folders' self.categories = ['birthday', 'anniversary', 'love', 'valentine',
'wedding'] self.effects = Effects() self.transitions = Transitions() def
select_folder(self): category = random.choice(self.categories) return
os.path.join(self.base_path, category), category def select_media(self, folder_path):
images = random.sample(os.listdir(folder_path), 10) return [os.path.join(folder_path, img)
for img in images] def select_song(self, folder_name): song_folder =
os.path.join(self.base_path, folder_name, 'songs') song_path = os.path.join(song_folder,
random.choice(os.listdir(song_folder))) return song_path def
apply_effects_and_transitions(self, image_clips): # Apply random effects and transitions
for i, clip in enumerate(image_clips): # Apply random effect effect_method =
random.choice([method for method in dir(self.effects) if callable(getattr(self.effects,
method)) and not method.startswith("__")]) clip = getattr(self.effects, effect_method)
(clip) # Apply random transition if not the first image if i > 0: transition_method =
random.choice([method for method in dir(self.transitions) if
callable(getattr(self.transitions, method)) and not method.startswith("__")]) clip =
getattr(self.transitions, transition_method)(clip) image_clips[i] = clip return
image_clips def create_video(self): folder_path, category = self.select_folder() images =
self.select_media(folder_path) song_path = self.select_song(category) image_clips =
[ImageClip(m).set_duration(6) for m in images] # Set duration for each image image_clips =
self.apply_effects_and_transitions(image_clips) video =
concatenate_videoclips(image_clips, method="compose") # Adjust song length song =
AudioFileClip(song_path) if song.duration > 60: song = song.subclip(0, 60) # Trim to 60
seconds elif song.duration < 60: song = song.loop(duration=60) # Loop to 60 seconds
final_video = video.set_audio(song) output_path = f"./Generated
video/{category}/{category}_video.mp4" final_video.write_videofile(output_path) return
output_path def upload_to_youtube(self, video_path, category): # Placeholder for Selenium
YouTube upload script pass # Usage: video_creator = VideoCreator() video_path =
video_creator.create_video() video_creator.upload_to_youtube(video_path,
video_creator.select_folder()[1])
```

This code provides the structure for the library. You would need to fill in the Selenium logic for YouTube uploading, as that would depend on your web browser, YouTube's current layout, and personal account information.

Please ensure that the paths, environment settings, and dependencies are correctly set up for your specific project and system.

Notes:

The **`Effects`** and **`Transitions`** classes are placeholders. You should fill them with actual effect and transition methods.

The **`upload_to_youtube`** method in **`VideoCreator`** is also a placeholder. You need to implement the Selenium logic according to your requirements.

Let me know if there are specific parts you want to expand or if there's anything else you need!

Hotkeys:

W: Yes, Continue
A: Alt
S: Explain
D: Iterate, Improve, Evolve

You
W

Grimoire
Great, let's continue with the next steps. If you've followed the structure correctly, you should now have a Python library capable of generating videos from images and songs based on specific categories. These videos will have random effects and transitions applied to them. Also, you should have a skeleton function ready for uploading videos to YouTube using Selenium.

Now, I'll provide more detailed steps for what could be your next moves:

Expand Effects and Transitions: Currently, the `Effects` and `Transitions` classes have placeholders. You should implement real image effects and transitions. This could involve using libraries like OpenCV or PIL for image manipulation, and MoviePy for video transitions.
Finalize Video Creation: Make sure the video creation logic correctly combines images, applies effects and transitions, and adds background music. Test this part thoroughly with different sets of images and songs to ensure the videos are generated as expected.
Implement Selenium Automation for YouTube: This involves navigating to YouTube, logging in, uploading the video, and filling out the title, description, and tags. This part will require familiarity with Selenium and web automation. Note that YouTube's layout and elements may change over time, so this code may need updates in the future.
Testing and Debugging: Test the entire library with different categories and ensure all parts are working together seamlessly. Handle exceptions and errors that may occur, especially in file handling and web automation.
Documentation and Cleanup: Write clear documentation for your library, explaining how to install it, how to use it, and what each part of the code does. Clean up the code to improve readability and performance.
Optional Features: Consider adding additional features based on your needs, such as support for more image and song formats, more complex effects and transitions, or more advanced YouTube upload options.

If you need further assistance with any of these steps, specific code examples, or explanations, feel free to ask!

Hotkeys:

E: Expand
T: Test cases
F: Fix. Code didn't work
H: help. debug lines

You
1 and 3 Expand , specific code examples

Grimoire
Alright, let's expand on both the effects and transitions for the videos and the Selenium automation for YouTube uploads.

1. Expanding Effects and Transitions

For the effects and transitions, we'll use the MoviePy library, which provides a good range of options. I'll add a few example methods to the `Effects` and `Transitions` classes. Note that you should have MoviePy installed (`pip install moviepy`) and you may need additional libraries for more advanced effects.

```
from moviepy.editor import VideoFileClip, concatenate_videoclips, vfx class Effects: def
black_and_white(self, clip): """Converts clip to black and white.""" return
clip.fx(vfx.blackwhite) def mirror_x(self, clip): """Mirrors the clip horizontally."""
return clip.fx(vfx.mirror_x) def invert_colors(self, clip): """Inverts the colors of the
clip.""" return clip.fx(vfx.invert_colors) # You can add more effects here class
Transitions: def make_transition(self, clip1, clip2, duration=1): """Creates a simple
crossfade transition.""" return concatenate_videoclips([clip1, clip2], method="compose",
padding=-duration).set_duration(clip1.duration + clip2.duration - duration) # Add more
transitions here
```

In this example, I added a few basic effects using MoviePy's `vfx` module and a simple transition method that creates a crossfade between two clips. You can add more effects and transitions based on the MoviePy documentation and your needs.

3. Implementing Selenium Automation for YouTube Uploads

To automate YouTube video uploads with Selenium, you'll first need to install Selenium and download a WebDriver for your browser. I'll use Chrome in this example, but you can adjust it according to your browser.

First, make sure you have the necessary components:

bash Copy code

```
pip install selenium
```

Download the ChromeDriver from the official site and ensure it matches your Chrome version.

Here's a basic outline of how the automation might look:

python Copy code

```
from selenium import webdriver from selenium.webdriver.common.by import By from
selenium.webdriver.common.keys import Keys from selenium.webdriver.support.ui import
WebDriverWait from selenium.webdriver.support import expected_conditions as EC class
VideoCreator: # Previous methods... def upload_to_youtube(self, video_path, category):
title = f"{category.capitalize()} Video" description = f"This is a beautiful {category}
video created automatically." tags = f"{category}, video, automatic" driver =
webdriver.Chrome('/path/to/chromedriver') # Update path as needed
driver.get('https://www.youtube.com/upload') # Log in to Google account # Note: This step
might require handling 2FA or CAPTCHAs manually WebDriverWait(driver,
20).until(EC.element_to_be_clickable((By.NAME,
"identifier"))).send_keys('your_email@gmail.com', Keys.ENTER) WebDriverWait(driver,
20).until(EC.element_to_be_clickable((By.NAME, "password"))).send_keys('your_password',
Keys.ENTER) # Wait for the upload page to load and upload video WebDriverWait(driver,
30).until(EC.element_to_be_clickable((By.CSS_SELECTOR,
"input[type='file']"))).send_keys(video_path) # Set video details WebDriverWait(driver,
20).until(EC.element_to_be_clickable((By.ID, 'textbox'))).send_keys(title)
driver.find_elements_by_id('textbox')[1].send_keys(description)
driver.find_elements_by_id('textbox')[2].send_keys(tags) # You might need to adjust the
selectors based on YouTube's current layout # This is a simplified version; YouTube's
interface may require handling more elements # Submit the video (Commented out to prevent
accidental uploads during tests) # WebDriverWait(driver,
20).until(EC.element_to_be_clickable((By.ID, "done-button"))).click() # Close the browser
driver.quit()
```

Please note:

This is a simplified version. YouTube's UI changes frequently, so you may need to update the selectors.
Automating Google login might trigger CAPTCHA or 2FA; handling these programmatically is complex and against Google's policies. You might need to log in manually or find alternative methods for authentication.
The actual upload process may include more steps based on YouTube's current interface, like setting video visibility.

Remember, automation of YouTube uploads should comply with YouTube's terms of service. Automated uploads can lead to issues with your account, including potential bans if misused or overused.

Hotkeys:

- **T**: Test cases
- **F**: Fix. Code didn't work
- **H**: help. debug lines
- **D**: Iterate, Improve, Evolve

You

You have mssign the promt part which is :

However, the logic for selecting the appropriate title, description, and tags based on the type of video needs to be implemented.

Grimoire

Let's address the logic for dynamically generating titles, descriptions, and tags for the YouTube video based on the selected category. This part of the code will be implemented within the `upload_to_youtube` method of the `VideoCreator` class.

The idea here is to have pre-defined title, description, and tag templates for each category (e.g., birthday, anniversary, etc.) and then fill these templates with relevant information when uploading the video. This approach ensures that the uploaded video content is consistent with the selected category.

Below is an extension of the `upload_to_youtube` method, incorporating logic for selecting the appropriate title, description, and tags based on the video category:

```python
def upload_to_youtube(self, video_path, category): # Define templates for titles,
descriptions, and tags video_details = { 'birthday': { 'title': 'Unforgettable Birthday
Memories | Created with Love', 'description': 'Dive into the joy and celebration of
unforgettable birthday moments with this special video.', 'tags': 'birthday, celebration,
joy, happy birthday' }, 'anniversary': { 'title': 'Happy Anniversary | Cherishing Moments
Together', 'description': 'Celebrating the journey of love and togetherness. Happy
Anniversary!', 'tags': 'anniversary, love, celebration, together' }, 'love': { 'title':
'Expressions of Love | Heartfelt Moments', 'description': 'A beautiful compilation
expressing the profound and beautiful moments of love.', 'tags': 'love, romance,
affection, heartfelt' }, 'valentine': { 'title': 'Valentine\'s Day Special | Love in the
Air', 'description': 'Celebrate the essence of love this Valentine\'s Day with our special
video.', 'tags': 'valentine, love, romance, valentine\'s day' }, 'wedding': { 'title':
'Wedding Bliss | A New Beginning', 'description': 'Embrace the journey of love and new
beginnings with our wedding video.', 'tags': 'wedding, love, marriage, new beginnings' } }
# Select details based on category details = video_details.get(category,
video_details['love']) # Default to 'love' if category not found driver =
webdriver.Chrome('/path/to/chromedriver') # Update path as needed
driver.get('https://www.youtube.com/upload') # Log in and upload video steps... # Set
video details WebDriverWait(driver, 20).until(EC.element_to_be_clickable((By.ID,
'textbox'))).send_keys(details['title']) driver.find_elements_by_id('textbox')
[1].send_keys(details['description']) driver.find_elements_by_id('textbox')
[2].send_keys(details['tags']) # Remaining upload steps...
```

In this example, `video_details` is a dictionary where each key is a video category and each value is another dictionary containing the title, description, and tags specific to that category. When uploading a video, the script looks up this dictionary to find the appropriate details based on the video's category.

This approach provides a structured way to manage video metadata and ensures that the content you upload is relevant to the video's theme. Of course, you can expand or modify the templates based on your needs or the specifics of your content.