oki

Report Videos Check

COMP1752 \_ CoUrsework

Nguyen Thanh Vinh

2023

Contents

[1. Basic understanding 2](#_Toc152884863)

[2. Design 3](#_Toc152884864)

[2.1. Create\_video\_list.py 4](#_Toc152884865)

[3. Implementation 7](#_Toc152884866)

[3.1. Create\_video\_list.py 7](#_Toc152884867)

[3.2. Update\_video.py 8](#_Toc152884868)

[4. Testing and Validation 10](#_Toc152884869)

[5. Innovations and Future Development 12](#_Toc152884870)

[5.1. Display video information when clicked video name in table: 12](#_Toc152884871)

[5.1.1. Get clicked index 12](#_Toc152884872)

[5.1.2. Determine Line Number 12](#_Toc152884873)

[5.1.3. Get Video Number: 12](#_Toc152884874)

[5.1.4. Retrieve Video Information 12](#_Toc152884875)

[5.1.5. Display Video Information 13](#_Toc152884876)

[5.1.6. Update Status Label 13](#_Toc152884877)

[5.2. Show the most rating video in update\_video.py 13](#_Toc152884878)

[5.2.1. Find the most rating 13](#_Toc152884879)

[5.2.2. Method ‘show\_most\_rating\_video’: 14](#_Toc152884880)

[Conclusion 14](#_Toc152884881)

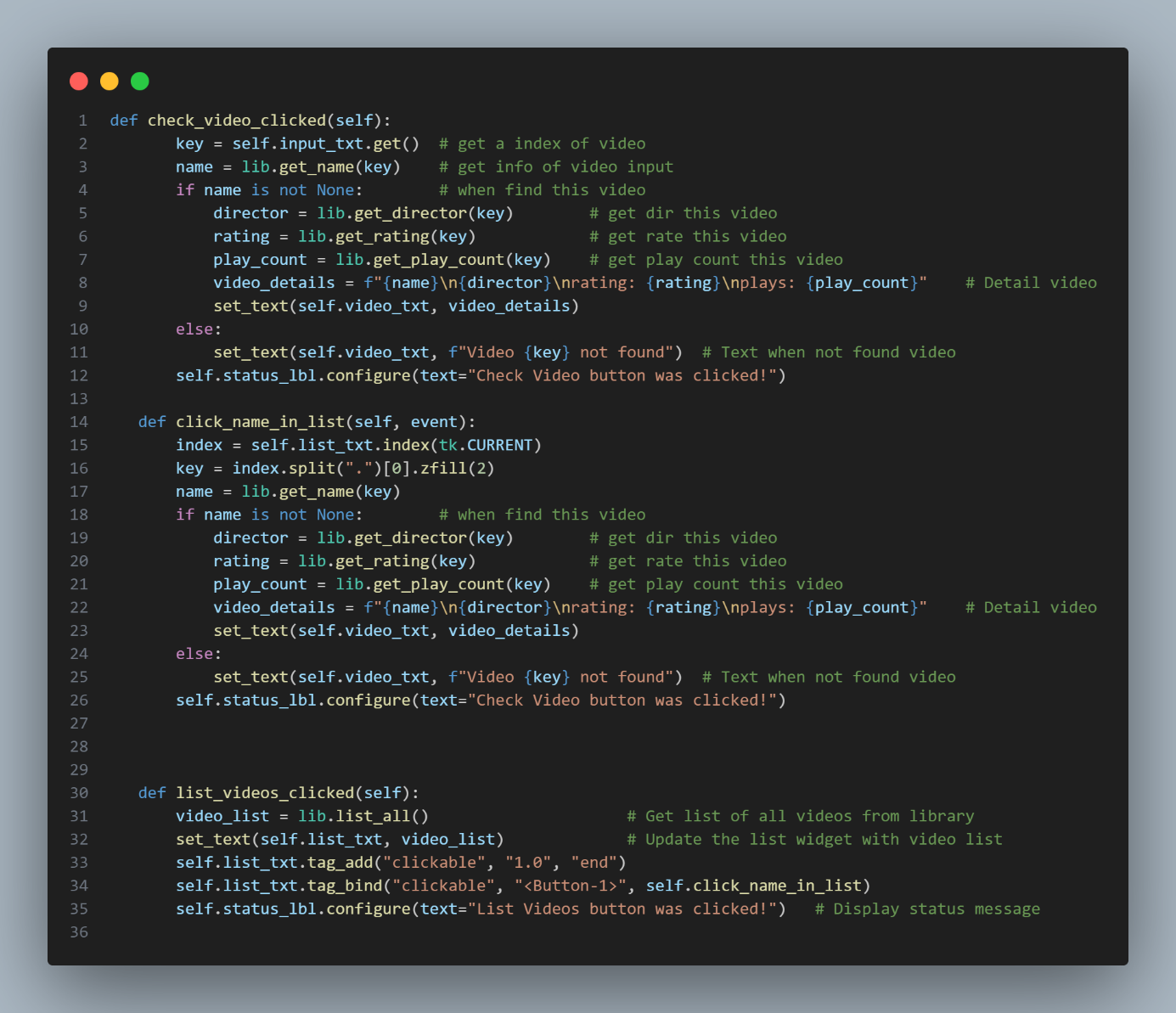
# Basic understanding

In this stage, the **check\_video.py** script was thoroughly reviewed and documented. Each function’s purpose and functionality were explained with comments. For example, the **set\_text** function was documented as follows:

def set\_text(text\_area, content):   #Set text when open label

    text\_area.delete("1.0", tk.END) #Delete previous content

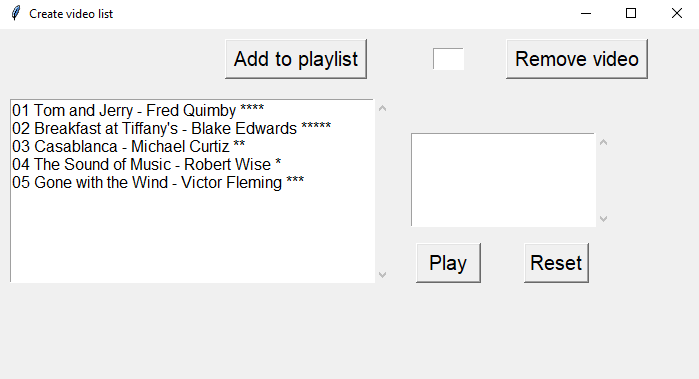
    text\_area.insert(1.0, content)  #Insert new content



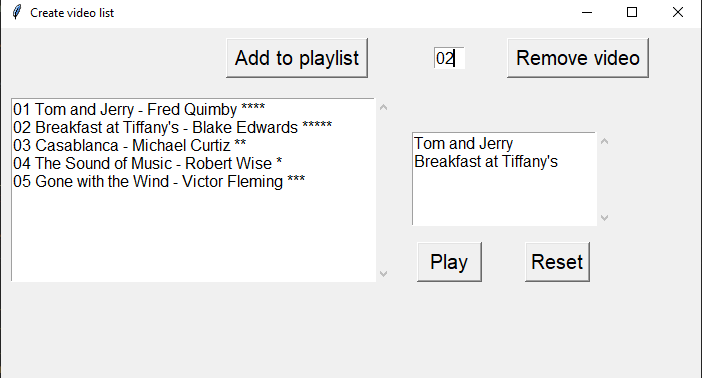
# Design

Video Player Simulation project aims to create a user-friendly video player application that emulates the behavior of a real video player. This report outlines the design decisions, architectural considerations, and key features implemented in the video player. This project has four guis include video\_player.py, check\_video.py, create\_video\_list.py, and update\_video.py. In particular, the module provides video\_played.py and check\_video.py.

## Create\_video\_list.py

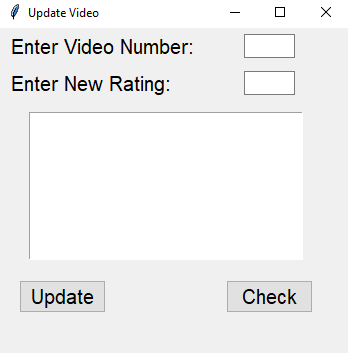
Create video list have three main parts included table list all video, enter the number, add to playlist and the last is play playlist and reset playlist. Firstly, the table displays the list of videos that can be added to the playlist, which is the biggest in the GUi. This table can be found beneath the area where you enter a number and select "Add to playlist."

**Figure 1: Create video list GUI**

Following the addition of videos, there is an additional, smaller panel to view playlist. This table is to the right of the table that shows all the videos.

Placing the entry field to input video numbers and the "Add to playlist" button at the top of the widget contributes significantly to a more user-friendly design. This arrangement aligns with users' natural reading patterns, ensuring that their attention is immediately drawn to the key elements upon entering the interface. Moreover, this design decision aligns with standard UI conventions, reducing cognitive load and providing a familiar experience for users. Then, by locating "Remove video" button next to the "Add to playlist" button, users can easily control their playlist in a single, continuous flow while still maintaining a logical and sequential structure. This feature gives the user more control over the playlist's content by providing an easy-to-use method for removing undesired movies. In conclusion, the addition of the "Remove video" option improves the user experience by enabling quick and simple playlist customization. Lastly, the “Play” button and “Reset” button located under the table show video playlist. the "Reset" button positioned alongside the "Play" button provides users with a straightforward means to clear and reset the playlist, ensuring a quick and accessible method for starting anew. This arrangement not only adheres to common user interface conventions but also streamlines the user's interaction with playlist management, contributing to an overall user-friendly design.

* 1. Update\_video.py

The update video widget has three main parts, including the entry to the video number and new rating, the table shows the information about the video, the “Update” button, and the “Check” button. The entry of the number video and new rating at the top of the widget is a key aspect of creating a more user-friendly design. This design enhances the user experience by ensuring that the primary input elements are immediately visible and easy to use.

The table serves as a helpful checkpoint by offering a preview of the information preceding and following the update. This lowers the possibility of mistakes and gives users peace of mind that their intended changes will be appropriately reflected. This design not only facilitates a more intuitive update process but also prioritizes user awareness and precision in managing video details. Overall, the table's role in presenting information before and after the update contributes significantly to the overall user-friendly design of the application. the placement of the "Update" and "Check" buttons under the table optimizes user interaction by providing quick access to essential actions related to video updates and information verification. This design choice contributes to a user-friendly interface, making the application more intuitive and efficient for users to navigate.

# Implementation

## Create\_video\_list.py

This outline provides a structured approach to building a video playlist application with Tkinter, featuring the ability to add, remove, play, and reset videos in the playlist. The methods are organized to handle different aspects of the application's functionality.

1. **Initialization:**

* The \_\_init\_\_ method initializes the main attributes and sets up the initial state of the application.
* The window is configured with a specific geometry and title
* **self.playlist** is a list that will store the selected videos for the playlist.

1. ScrolledText Widgets:

* Two ScrolledText widgets (**self.list\_txt** and **self.playlist\_text**) are created to display the list of all videos and the playlist, respectively.
* The tag\_configure method is used to configure a tag named 'clickable' for later use.

1. **Entry and Buttons:**

* An Entry widger (‘**self.video\_number\_entry**’) is created to input the video number
* Two buttons (‘**self.btn\_add\_video’** and ‘**self.btn\_remove\_video’**) are created to add and remove videos from the playlist, respectively
* Two more buttons (‘self.btn\_play\_video’ and ‘self.btn\_reset\_video’) are create to play the playlist and reset it

1. **Method ‘add\_video’:**

* This method retrieves the video number from the entry widget (‘**video\_number\_entry**’).
* It uses the lib.get\_name function to obtain the corresponding video name.
* If the video name is valid (i.e., not ‘**None**’), it adds the video to the playlist, and the playlist display is updated using ‘**update\_playlist\_text**’.
* If the video number is invalid, it displays an error message.

1. Method ‘play\_video’:

* This method takes a ‘key’ in the ‘video\_store’ and uses the ‘lib.increment\_play\_count’ function to increment the play count for the playlist
* Appear a messagebox to notification video play successfully

1. **Methods:**

* ‘**list\_videos\_clicked**’: Fetches the list of all videos from the library (‘**lib.list\_all()**’) and displays it in the **self.list\_txt** widget using the set\_text function.
* ‘**update\_playlist\_text**’: Updates the **self.playlist\_text** widget with the current playlist content and binds the "clickable" tag to each video for future interaction.
* ‘**display\_error\_message’**: Displays an error message label with the given message on the window.
* ‘**remove\_video’**: Removes a video from the playlist based on the input video number.
* **‘add\_video\_clicked’**: Handles the event when a video in the list is clicked. Adds the corresponding video to the playlist and updates the UI.
* **‘reset\_playlist’**: Clears the playlist and updates the UI.

1. **Main Block:**

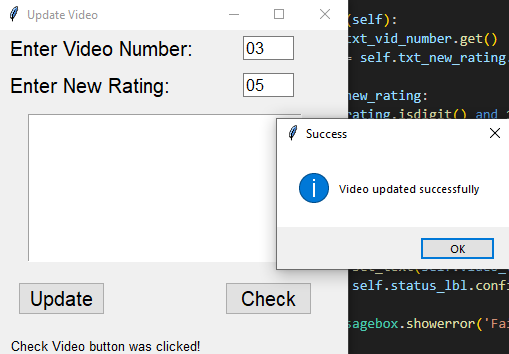
* The ‘if \_\_name\_\_ == “\_\_main\_\_” ‘: block creates a Tkinter window (‘**window**’), configures fonts, initializes the ‘**CreateVideo**’ class, and starts the Tkinter event loop with ‘**window.mainloop()**

## Update\_video.py

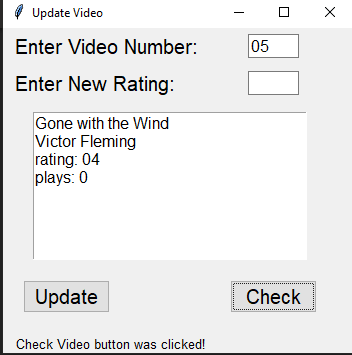
1. **Initialization:**

* The ‘**\_\_init\_\_**’ method sets up the initial state of the update video window.
* The window is configured with a specific geometry and title.
* Labels and entry widgets are created to input the video number and new rating.
* A Text widget (‘**self.video\_txt’**) is used to display video information.
* Buttons for updating and checking video information are created.

1. **Method ‘update\_rate’:**

* This method is called when the "**Update**" button is clicked.
* It retrieves the video number and new rating from the entry widgets.
* If both fields are non-empty, it checks if the video exists using **lib.get\_rating(key)**
* If the video exists, it updates the rating using **lib.set\_rating(key, new\_rating)**.
* A success message is shown using **messagebox.showinfo**
* The video number and new rating entry fields are cleared, and the video text is reset.
* If the video is not found, an appropriate message is displayed.

1. **Method ‘check\_video\_clicked’:**

* This method is called when the "**Check**" button is clicked.
* It retrieves the video number from the entry widget.
* If the video number is non-empty, it retrieves the video details (name, director, rating, play count) using library functions.
* The video details are formatted and displayed in the ‘**self.video\_txt**’ Text widget.
* If the video is not found, an appropriate message is displayed
* The status label (‘**self.status\_lbl**’) is updated with a message indicating that the "**Check Video**" button was clicked.

1. **Main block:**

* The ‘**if \_\_name\_\_** == “**\_\_main\_\_**” ’: block create a Tkinter window (‘**window**’), configures fonts, initializes the ‘**UpdateVideo**’ class, and startss the Tkinter event loop with ‘**window.mainloop()**’

# Testing and Validation

* 1. create\_video\_list.py

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Test | Purpose | Input | Expected |
| 1 | Add the playlist | Add video to playlist | Video Number: 01  Click “Add to playlist” button | Display video name in playlist |
| 2 | Add invalid video number | Add Invalid Video to Playlist | Video Number: 999 | Error: Invalid video number |
| 3 | Remove video from playlist | Remove Video from Playlist | Add Video 01  Enter Video Number: 01  Click “Remove Video” button | Video should be removed from the playlist |
| 4 | Play playlist | Play Playlist | Add video 01  Click “Play” button | Play count for Video 01 should be incremented |
| 5 | Reset playlist | Reset Playlist | Click "Reset" button | Playlist should be cleared |

* 1. update\_video.py

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Test | Purpose | Input | Expected |
| 1 | Test correct value | Valid Update | Video Number: 01  New Rating: 04 | Video updated successfully |
| 2 | Unvalid | Unvalid Update | Video Number: 01  New Rating: 06 | Error: The new rating must be an integer between 1 and 5 |
| 3 | Video not found | Video not found | Video Number: 06 | Video 06 not found |
| 4 | Show Most Rating Video | Click "Show Most Rating Video" button | Display details of the video with the highest rating | Show Most Rating Video |
| 5 | Empty Fields | Click "Update" button without entering values | Please enter video number and new rating | Empty Fields |

# Innovations and Future Development

## Display video information when clicked video name in table:

### Get clicked index

* ‘index = self.list\_txt.index(tk.CURRENT)’: Retrieves the index of the clicked position in the scrolled text widget (list\_txt).
* The tk.CURRENT represents the current position of the mouse cursor.

### Determine Line Number

* ‘**line\_number = int(index.split(".")[0])**’: Extracts the line number from the index.
* This line number corresponds to the line where the user clicked.

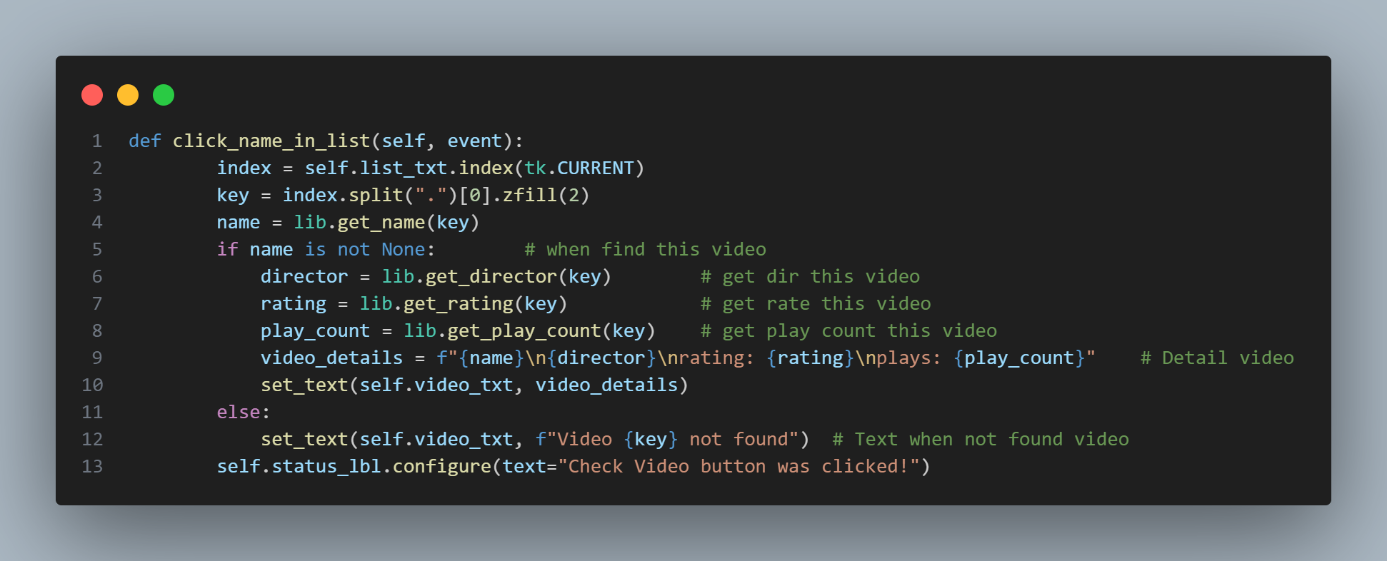
### Get Video Number:

* ‘**video\_number = str(line\_number).zfill(2)**’: Converts the line number to a video number (assumed to be a two-digit format).

### Retrieve Video Information

* ‘**name = lib.get\_name(video\_number)**’: Retrieves the name of the video from the video library.
* ‘**director = lib.get\_director(video\_number)**’: Retrieves the director of the video.
* ‘**rating = lib.get\_rating(video\_number)**’: Retrieves the rating of the video.
* ‘**play\_count = lib.get\_play\_count(video\_number)**’: Retrieves the play count of the video.

### Display Video Information

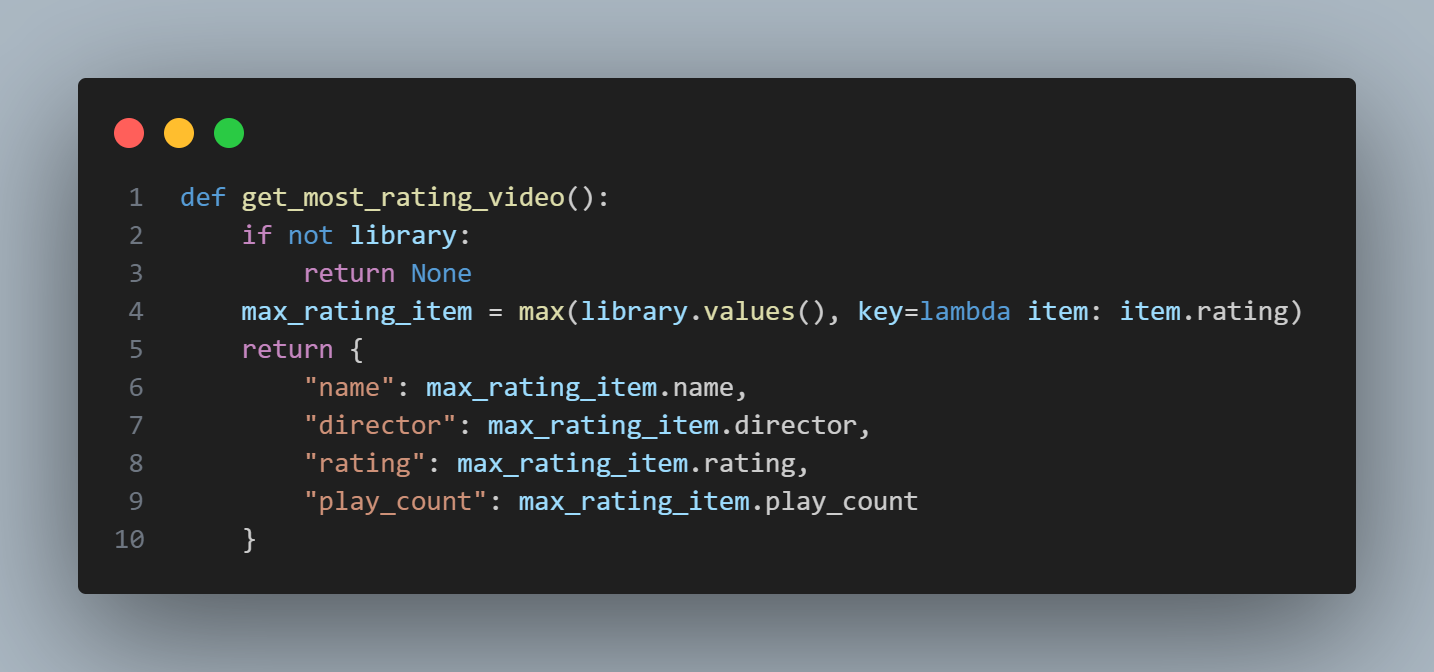
* If the video is found (‘**name is not None**’), it constructs a string (‘**video\_details’**) containing various details such as name, director, rating, and play count.
* **‘set\_text(self.video\_txt, video\_details)’**: Sets the content of the ‘**video\_txt’** text widget to display the video details.

### Update Status Label

* ‘**self.status\_lbl.configure(text="Video name clicked!")**’: Updates a label (‘**status\_lbl**’) to indicate that a video name has been clicked.

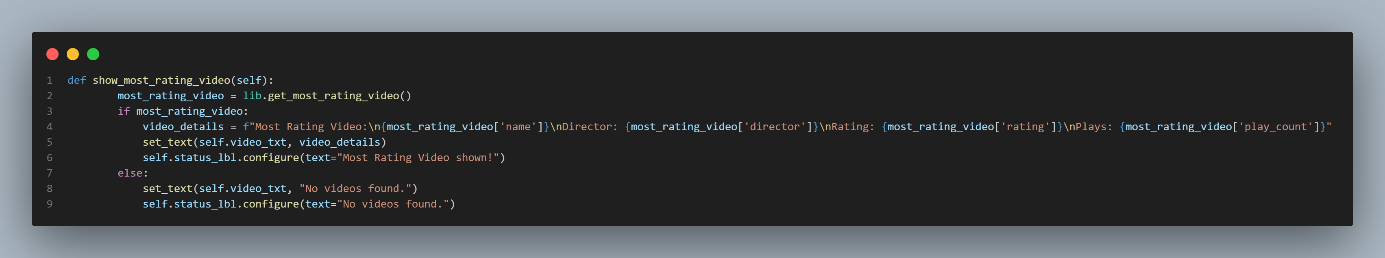
## Show the most rating video in update\_video.py

### Find the most rating

* **‘**if not library**’**: This condition checks if the library dictionary is empty. If it is, meaning there are no items in the library, the function returns None because there is no most-rated video to retrieve**.**
* ‘**max\_rating\_item = max(library.values(), key=lambda item: item.rating)**’: This line uses the max function to find the item in the library dictionary with the maximum rating. The key parameter is set to a lambda function that extracts the rating attribute of each **LibraryItem** in the **library.values().**
* ‘**return{}**’: This line uses to get information of the most video rating.

### Method ‘show\_most\_rating\_video’:

* most\_rating\_video = lib.get\_most\_rating\_video(): This line calls the get\_most\_rating\_video function from the lib module to retrieve the key of the most-rated video in the library.
* if most\_rating\_video:: Checks if most\_rating\_video is not None, meaning there is a most-rated video in the library.
* self.status\_lbl.configure(text="Most Rating Video shown!"): Updates the status label to indicate that the most-rated video has been shown.



# Conclusion

This coursework was extremely helpful in the growth of my knowledge and abilities in Python programming proficiency, GUI development, and Object-Oriented Programming (OOP). My Python programming abilities were able to be applied and improved upon thanks to the coursework. Moreover, I created a GUI using Tkinter and expanded my skill in designing user-friendly interfaces, handing user inputs, and connecting ìnterfaces elements with underlying functionality. And lastly, OOP abilities also significantly enhance following curriculum completion.