T1A3

Chris lee

My terminal app

My terminal app is a horror theme virtual movie program that allows users to search its database, add to the database, edit the ratings of movies, delete movies from the database and then allows them to rent a movie.

The main 3 features of my terminal app are (in no particular order):

-1: Edit movie ratings

-2: Edit movie list

-3: Rent a movie

Feature 1: The first feature allows users to edit the rating of a movie. This can either be one that's already in the list or one that they have added. When called, this function prints the movie list and prompts the user to enter the name of the movie they'd like to change the rating for.

```
chris@DESKTOP-QRQO39E:~/T1A3$ python3 main2.py
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6): 3
Welcome to our Movie Database. These are our options:
The Haunting: Rating: 5
The Ritual: Rating: 4
Necromancers: Rating: 5
Kidnapping: Rating: 3
Slaughterhouse: Rating: 2
Stalker: Rating: 1
Whats the name of the movie who's rating youd like to edit?:
```

Once they select a movie, they are prompted to provide the new rating. Afterwards, they can reprint the movie list if they like and the user will see the updated score.

```
Select an option (1-6): 3
Welcome to our Movie Database. These are our options:
The Haunting: Rating: 5
The Ritual: Rating: 4
Necromancers: Rating: 5
Kidnapping: Rating: 3
Slaughterhouse: Rating: 2
Stalker: Rating: 1
Whats the name of the movie who's rating youd like to edit?: Stalker
What is the updated rating of Stalker?: 5
Stalker's rating was updated!
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

Pictured below, is the updated score for the movie the user selected. (“Stalker”)

```
Select an option (1-6): 1
Welcome to our Movie Database. These are our options:
The Haunting: Rating: 5
The Ritual: Rating: 4
Necromancers: Rating: 5
Kidnapping: Rating: 3
Slaughterhouse: Rating: 2
Stalker: Rating: 5.0
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

Feature 2: The second feature for my terminal app allows the user to edit the movie list. This can be done via adding a movie to the list or deleting a movie from the list.

In this picture, the user added a movie called “super scary movie” to the database. They also provided a rating of 10 for this movie. After the movie name and rating is provided, reprinting the movie list displays the new movie list.

```
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6): 2
Whats the name of the movie youre adding?: Super Scary Movie
Whats the rating of Super Scary Movie?: 10
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6): 1
Welcome to our Movie Database. These are our options:
The Haunting: Rating: 5
The Ritual: Rating: 4
Necromancers: Rating: 5
Kidnapping: Rating: 3
Slaughterhouse: Rating: 2
Stalker: Rating: 5.0
Super Scary Movie: Rating: 10.0
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

Deleting a movie from the list is just as easy. Any movie can be deleted from the list, including movies that were predefined by me. In the picture to the right, the user deleted “Super Scary Movie”. The program prompted the user that the movie was successfully deleted and re printing the movie list shows this is true.

```
Select an option (1-6): 4
What is the name of the movie you're removing: Super Scary Movie
Super Scary Movie was removed from our database!
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6): 1
Welcome to our Movie Database. These are our options:
The Haunting: Rating: 5
The Ritual: Rating: 4
Necromancers: Rating: 5
Kidnapping: Rating: 3
Slaughterhouse: Rating: 2
Stalker: Rating: 5.0
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

Feature 3: The third features pertains to renting a movie. This includes movies users add as well as the predefined list of movies. Pictured right, users can select a movie from the list that they want to loan. They can loan the movie for up to 19 days. Anything longer is denied by the program.

```
chris@DESKTOP-QRQO39E:~/T1A3$ python3 main2.py
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6): 5
Welcome to our Movie Database. These are our options:
The Haunting: Rating: 5
The Ritual: Rating: 4
Necromancers: Rating: 5
Kidnapping: Rating: 3
Slaughterhouse: Rating: 2
Stalker: Rating: 1
Whats the name of the movie youre borrowing?: Stalker
How long (# in days) are you borrowing Stalker for?: 5
Stalker: is loaned to you for: 5.0 days! Enjoy
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

Code/Terminal app logic: Throughout the program, i utilise simple *if* statements and *for* loops to define logic and conditional actions. For example, below is the rent a movie function. Considering the broad scope of what a movie name can be/include, i opted to target empty entries from the user when prompting them for a movie name. The try/except block comes into play when input criteria is reduced, as in, the rating of a movie can only be a number/decimal. Anything else raises a value error.

You may notice as well, the “Back.Yellow +” text inside the print statements. This is from an imported module used to “colorize” the terminal output.

```
def rent_movie():
    max_length = 20
    movie_menu.print_movie_menu()
    name=input(Back.LIGHTBLUE_EX +"Whats the name of the movie youre borrowing?: ")
    if name == "":
        print(Back.YELLOW +"Invalid option, empty entries are not allowed.")
    try:
        length = float(input(Back.LIGHTBLUE_EX +f"How long (# in days) are you borrowing {name} for?
        : "))
        if length >= max_length:
            print(Back.YELLOW +"Maximum rental period is 19 days.")
        if length == "":
            print(Back.YELLOW +"Invalid option, empty entries are not allowed.")
        else:
            if length <= 19:
                print(f"{name}: is loaned to you for: {length} days! Enjoy")
    except ValueError:
        print(Back.YELLOW +"Value Error. Rental period input must be integer.")
```

Overall structure.

The main file for the app executes a few print statements to begin. There is a few functions defined here as well, with the remaining functions/classes being imported from other local files. These functions rely on classes defined elsewhere. On the next slide, ill show the last chunk of the main file.

OPEN EDITORS
- run_movie.sh
- seed.py
- main2.py
- rent.py
- readme.md M
- Movie_menu_ite...
- Movie_Menu.py

T1A3 [...
- __pycache__
- main2.py
- Movie_menu_items.py
- Movie_Menu.py
- readme.md M
- rent.py
- run_movie.sh
- seed.py

main2.py > rent_movie

```python

movie_menu = movie_seed()

#----------------------------------------MAIN MENU FOR TERMINAL APP----------------------------------------------------------#
def print_options():
    print(Back.CYAN +"1.Show movie list")
    print(Back.CYAN+"2.Add a movie to our database")
    print(Back.CYAN+"3.Edit the rating of a movie on our database")
    print(Back.CYAN+"4.Delete a movie from our database")
    print(Back.CYAN+"5.Rent a movie from our collection")
    print(Back.CYAN +"6.Exit")
    option = input("Select an option (1-6): ")
    return option

#----------------------------------------END MAIN MENU-----------------------------------------------------------------------#
#----------------------------------------DEFINE MENU OPTIONS-----------------------------------------------------------------#

def add_movie():
    try:
        name=str(input(Back.LIGHTBLUE_EX +"Whats the name of the movie youre adding?: "))
        if name == "":
            print(Back.YELLOW +"Empty name entries are invalid. You will be required to rebsumit your entry.")
    except ValueError:
        print("ValueError")
    try:
        rating = float(input(Back.LIGHTBLUE_EX +f"Whats the rating of {name}?: "))
        movie_menu.add_movie(name,rating)
    except ValueError:
        return print(Back.YELLOW +"Invalid input type. Rating can only be a number. You will be required to resubmit your entry.")



def edit_rating():
    movie_menu.print_movie_menu()
    name = input(Back.LIGHTBLUE_EX +"Whats the name of the movie who's rating youd like to edit?: ")
    movie_menu.edit_movie_rating(name)


def remove_movie():
        name = input(Back.LIGHTBLUE_EX +"What is the name of the movie you're removing: ")
        movie_menu.delete_movie(name)


def rent_movie():
    max_length = 20
    movie_menu.print_movie_menu()
    name=input(Back.LIGHTBLUE_EX +"Whats the name of the movie youre borrowing?: ")
    if name == "":
        print(Back.YELLOW +"Invalid option, empty entries are not allowed.")
    try:
        length = float(input(Back.LIGHTBLUE_EX +f"How long (# in days) are you borrowing {name} for?: "))
        if length >= max_length:
            print(Back.YELLOW +"Maximum rental period is 19 days.")
        if length == "":
            print(Back.YELLOW +"Invalid option, empty entries are not allowed.")
        else:
            if length <= 19:
                print(f"{name}: is loaned to you for: {length} days! Enjoy")
    except ValueError:
        print(Back.YELLOW +"Value Error. Rental period input must be integer.")
```

This while loop is used to control user interaction with the app. To begin, the option is set to an empty string/no input. *If* statements are used to determine what input the user has selected. For example, *if* the user enters “2”, the add_movie() function is invoked. Here, input must be a numerical number, any other entry will active the “else” clause and alert the user that the program did not understand that input type.

```
option = ""
while option != "6":
    option = print_options()

    if option == "1":
        movie_menu.print_movie_menu()
    elif option == "2":
        add_movie()
    elif option == "3":
        edit_rating()
    elif option == "4":
        remove_movie()
    elif option == "5":
        rent_movie()
    elif option == "6":
        continue
    else : print(Back.YELLOW +"Sorry, i didnt understand that input type!")
print(Back.MAGENTA + Fore.WHITE +"Goodbye")
```

Trello boards: I used a trello board alongside the app.diagram website to create some basic app structure ideas and to begin plotting out how each part would communicate with itself and the user. The board has a variety of sections including scrapped ideas, main idea, main features etc.

The diagram website was used to plan my original app ideas flowchart, however, i scrapped that idea and am in the process of creating a new flowchart for this program you're seeing now.

Trello board can be accessed via this link:

https://trello.com/b/ikXsEPIs/terminal-app-t1a3

Google slide presentation can be accessed via this link:

https://docs.google.com/presentation/d/1PigqHKH3KcfWJ6ZvsX9q8nSfkDZAagziDBiGli0XWP4/edit#slide=id.g13d1a72dfcb_0_83

My github repository can be accessed via this link:

https://github.com/chrislee12189/T1A3

Error testing and handling results can be accessed via this link:

https://docs.google.com/spreadsheets/d/1efrQbJEYdCSuF6wbV7F3Qs6WEMpdF6MXvO5bObPIXcw/edit#gid=1342169021

Trello board:

Board | **Terminal App T1A3** | Trello Workspace | Public | Share | Power-Ups | Automation

App idea + 3 Features

- Online Movie Theatre
- Editable Movie list
- Editable Movie ratings
- Movie hire

+ Add a card

Functionality

- Main menu
- User input to travel through program
- Add movie (1)
- Edit rating (1)
- Remove movie (1)
- Rent movie (1)
- Exit program (1)

+ Add a card

Source control

- At least 20 commits

+ Add a card

Execution

- Need highly sophisticated scripts to facilitate execution of program

+ Add a card

Presentation

- Provide a very thorough walkthrough of the logic of the application.
- Slide deck (10 mins)
- Readme doc
- Readme doc needs to describe at least 3 features. Each to be explained individually

+ Add a card

Scrapped Ideas

- Magic 8 Ball style character creator
- Take user input/questionnaire
- Lists to assign user with random attributes, random dice roll generator style
- Lists containing character features ie class, strength, weakness

+ Add a card

Scrapped Functionality Planning

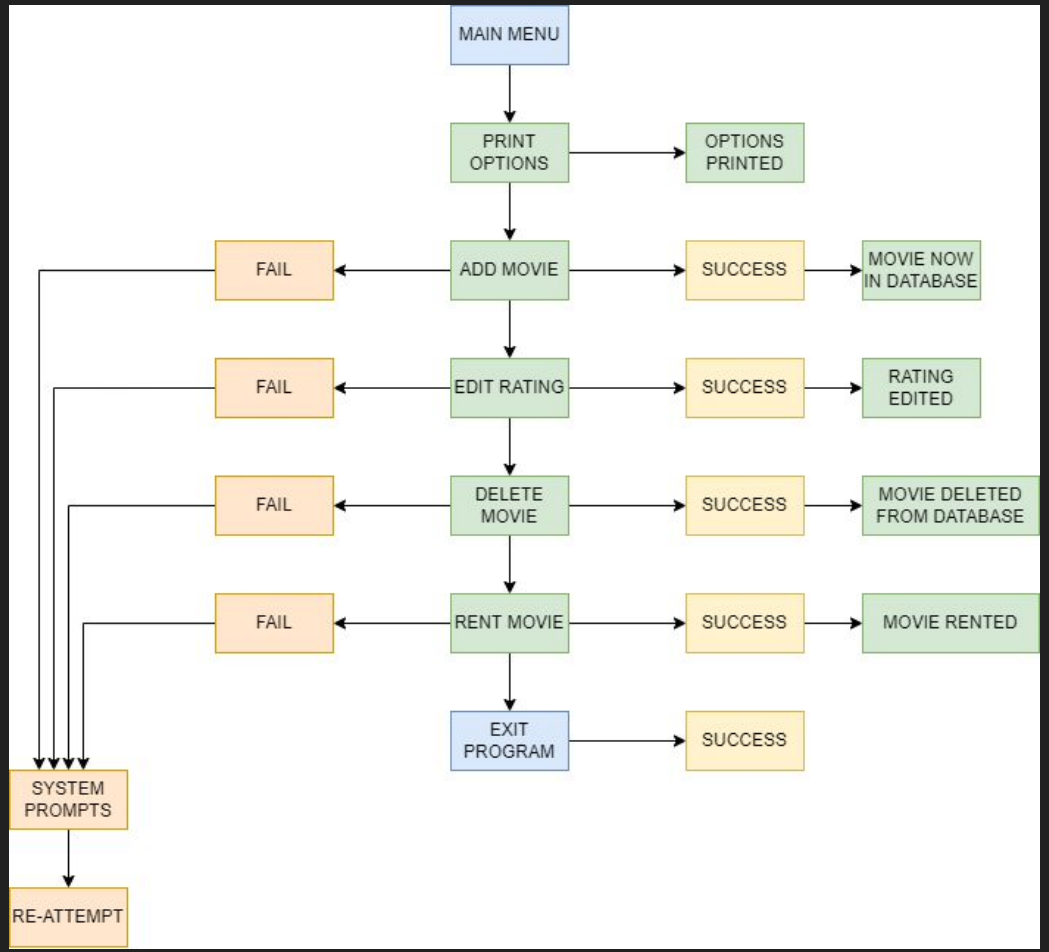
- Store user input in appended list. maybe use it somewhere? or just return information back to them as authentication style check
- Implement tests for app functionality.
- Use variables to store information that can be reused for output later on
- For/If/Elif/Else Loops for conditional results. i.e. return invalid inputs
- Import at least 4 python packages
- Make extensive use of functions from one or more python packages
- At least 6 functions, implement at least 5 of those
- Use input and output in 2 different ways each
- DRY code
- Application to run with 0 errors. Refer to Implement Tests card
- Help file to comprehensively instruct installation steps, dependencies, sys requirements and features of the application
- 2 tests to test main app functionality need to be highly relevant to checking if the application is running as expected

+ Add a card

Help file: There is a markdown file in my repository called “helpfile.md”. Its role is to give a user-friendly explanation of what the app is and how it works. It also includes a dependencies section with information about the library/module i use internally for the program. Installation instructions are located both in the readme.md and the helpfile.md

There are no minimum hardware requirements to run the app. There are some minor software requirements however, including the module “colorama” that i just spoke about, access to a terminal/bash shell/command prompt and internet access to clone my repository,

This flowchart was developed for my own needs and as such, does not follow standard flowcharting convention. I've included the picture as a means to further demonstrate my thought processes.



Code structure/ overview of critical application logic.

The app relies heavily on functions, classes and variables. The app also utilises a python package for styling.

The following slides will picture key code blocks for app functionality.

```python
movie_menu = movie_seed()

#----------------------------------------MAIN MENU FOR TERMINAL APP----
def print_options():
    print(Back.CYAN +"1.Show movie list")
    print(Back.CYAN+"2.Add a movie to our database")
    print(Back.CYAN+"3.Edit the rating of a movie on our database")
    print(Back.CYAN+"4.Delete a movie from our database")
    print(Back.CYAN+"5.Rent a movie from our collection")
    print(Back.CYAN +"6.Exit")
    option = input("Select an option (1-6): ")
    return option

#----------------------------------------END MAIN MENU----------------
```

The previous slide featured a global variable called “movie_menu”. This global variable gets its definition from an imported function called “movie_seed”. The “movie_seed” function contains more variables used to store movies that are predefined for the movie menu database. Without this, there would be no predefined movies for the user to browse upon running the application. The variables inside the “movie_seed” function contain the name of the movie and the integer rating of the movie.

Pictured right, are more functions used for interacting with the main menu when the program launches. After these functions are defined (they are not all defined just here, some functions were defined elsewhere and imported) they are called by *if* and *elif* statements.

Ie *if* option == 1:

add_movie()

```
#----------------------------------------DEFINE MENU OPTIONS----------------------------------------------------------------#

def add_movie():
    try:
        name=str(input(Back.LIGHTBLUE_EX +"Whats the name of the movie youre adding?: "))
        if name == "":
            print(Back.YELLOW +"Empty name entries are invalid. You will be required to rebsumit your entry.")
    except ValueError:
        print("ValueError")
    try:
        rating = float(input(Back.LIGHTBLUE_EX +f"Whats the rating of {name}?: "))
        movie_menu.add_movie(name,rating)
    except ValueError:
        return print(Back.YELLOW +"Invalid input type. Rating can only be a number. You will be required to resubmit your entry.")


def edit_rating():
    movie_menu.print_movie_menu()
    name = input(Back.LIGHTBLUE_EX +"Whats the name of the movie who's rating youd like to edit?: ")
    movie_menu.edit_movie_rating(name)


def remove_movie():
        name = input(Back.LIGHTBLUE_EX +"What is the name of the movie you're removing: ")
        movie_menu.delete_movie(name)


def rent_movie():
    max_length = 20
    movie_menu.print_movie_menu()
    name=input(Back.LIGHTBLUE_EX +"Whats the name of the movie youre borrowing?: ")
    if name == "":
        print(Back.YELLOW +"Invalid option, empty entries are not allowed.")
    try:
        length = float(input(Back.LIGHTBLUE_EX +f"How long (# in days) are you borrowing {name} for?: "))
        if length >= max_length:
            print(Back.YELLOW +"Maximum rental period is 19 days.")
        if length == "":
            print(Back.YELLOW +"Invalid option, empty entries are not allowed.")
        else:
            if length <= 19:
                print(f"{name}: is loaned to you for: {length} days! Enjoy")
    except ValueError:
        print(Back.YELLOW +"Value Error. Rental period input must be integer.")
```

The option menu functionality relies on a *while* loop. Nested *if* statements are used to check user input, if the input matches, the relevant function is called.

The *else* clause catches invalid inputs from the user. The option menu can only receive numeric numbers. I.e. “1” but not “One”

```
option = ""
while option != "6":
    option = print_options()

    if option == "1":
        movie_menu.print_movie_menu()
    elif option == "2":
        add_movie()
    elif option == "3":
        edit_rating()
    elif option == "4":
        remove_movie()
    elif option == "5":
        rent_movie()
    elif option == "6":
        continue
    else : print(Back.YELLOW +"Sorry, i didnt understand that input type!")
print(Back.MAGENTA + Fore.WHITE +"Goodbye")
```

Pictured right is one of the key classes for my terminal application. This class is defined in a separate file and then imported into the main file of the program. The purpose of using a class here is to reduce the amount of work it took to assign a new item with values. For example, adding a movie under the parameters of a Class allows me to easily add relevant information and have it stored correctly later. This allows me to take information from the user, categorize it, include it with predefined movies and further edit it later on.

The functions defined in this class were targeted by the functions on the main page (previous slides). These (class) functions intend to give far more power and flexibility to the main menu functions.

```
Movie_Menu.py > Movie_Menu > __init__
from Movie_menu_items import Movie_Menu_Items

#-------------------------------DEFINES FUNCTIONS THAT ARE ACCESSED FROM MAIN MENU-----------------------
class Movie_Menu:
    def __init__(self, movie_menu_items):
        self.movie_menu_items = movie_menu_items

    def print_movie_menu(self):
        print("Welcome to our Movie Database. These are our options: ")
        for item in self.movie_menu_items:
            item.show_item()


    def add_movie(self, name, rating):
        new_item = Movie_Menu_Items(name, rating)
        self.movie_menu_items.append(new_item)

    def delete_movie(self, name):
        for item in self.movie_menu_items:
            if item.name == name:
                self.movie_menu_items.remove(item)
                return print(f"{name} was removed from our database!")
        return print(f"{name} was not in the list!")

    def edit_movie_rating(self, name):
        for item in self.movie_menu_items:
            if item.name == name:
                try:
                    rating = float(input(f"What is the updated rating of {name}?: "))
                    item.rating = rating
                    return print(f"{name}'s rating was updated!")
                except ValueError:
                    return print("Value error caught. Integer is the only acceptable input for rating.")
        return print(f"{name} is not in our database!")
```

Another incredibly important aspect of code is the bash script picture right.

This bash script first checks the users computer for the required module and if they don't have it, installs it. After that, a simple message displays to the user for an optional help file to be printed. If they enter “y”, the help menu is printed. Otherwise, the script continues to execute the program.

$ run_movie.sh

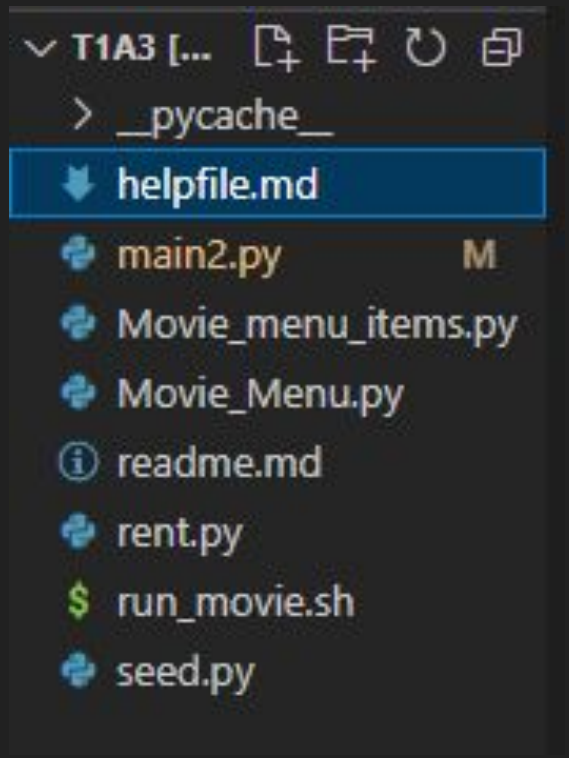
```bash
#!/bin/bash
pip freeze | grep 'colorama'
# python3 main2.py
echo 'Do you want to print the help file? (y/n)'
read helpfilereq
if [[ $helpfilereq == "y" ]];
    then
        cat helpfile.md \n
        python3 main2.py
    else
        python3 main2.py
fi
```

Pictured right, is the current state of my file tree. It will undergo changes before final submission. The purpose of this picture is to demonstrate local files and where the program is importing functions and classes from.

For example, “seed.py” contains the function which stores the variables for the predefined movie list.

“Movie_Menu.py” contains the class “Movie_Menu”, that is the class that helps provide extra capabilities to the main menu functions.

There are 2 document files, “readme.md” and “helpfile.md” both contain documentation pertaining to the development and functionality of the application.



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
chris@DESKTOP-QRQO39E:~/T1A3$ ./run_movie.sh
colorama==0.4.5
Do you want to print the help file? (y/n)
y
# **HELP FILE**
This terminal app was created for an assignment and features simple navigation capabilites, simple add/remove functionality and also a rental feature that allows the user to rent a movie.
Navigating the program is simple:
## MAIN MENU:
The app begins at the main menu. Users will be greeted with an options list. To select an option, users must enter a numerical number. i.e. "1" or "2" and not "one" or "two".
The options are:
- 1. Show movie list
- 2. Add a movie to the list
- 3. Edit the rating of a movie
- 4. Delete a movie from the database
- 5. Rent a movie from the list

## THE MENU IS PRINTED INCREDIBLY OFTEN AND USERS ARE ABLE TO INTERACT WITH IT AFTER ANY TASK COMPLETION, WHETHER THEY INPUT A VALID RESPONSE TO THEIR CURRENT ACTIVITY OR NOT.
### Option 1:
Option one prints the list of movies i pre defined for the program. If a user selects "2" from the main menu and correctly adds a movie, reprinting the list will show the original list and any movie they added. These entries are only valid for the current session, ending the program and rest
arting it will erase any additions/deletions the user makes.

### Option 2:
Option two allows the user to add a movie. They will be prompted for the name and the rating of the movie they want to add. In order to successfully add a movie, they need to provide a name (any str/integer is fine) and a rating (integer only). Failure to do so, the program will alert the us
er to this and the entry will **not** be added to the database.

### Option 3:
Option three allows the user to edit the rating of a movie. This can be either a movie theyve added (after successful addition of the movie) or a movie that is in the predefined list. Users will provide an integer to edit the rating.

### Option 4:
Option four allows the user to delete a movie from the database, like option 3, they can delete predefined movies or movies theyve added (after theyre added of course). To remove a movie, they must provide the name. This **is** case sensitive, ignoring case sensitivity will result in a faile
d attempt.

### Option 5:
Option five allows the user to rent a movie, the criteria to rent is: movie must be in the database, rental length must be less than or equal to 19 days. To rent a movie, the user is prompted for 2 inputs, movie name and length of rental. If the movie name entry is empty, the program alerts
the user and the attempt is invalid. If the rental length exceeds 19 days, it is also disallowed. A successful rental entry would be: correct in terms of case sensitivity and under the maximum number of days allowed for rent.

### Option 6:
Option six exits the program.



## IMPORTED LIBRARIES/MODULES
Documentation produced in the readme.md alerts users to the criteria needed for this program to function. I will place the same message here for ease of access:
# DEPENDENCIES/REQUIREMENTS/SYSTEM HARDWARE
- There is no minimum hardware requirements needed to run this app
- Link to colorama module: https://www.youtube.com/watch?v=u51Zjlnui4Y&ab_channel=TechWithTim
- In order for the colored text effects to work in terminal you will need install them. Installation is as easy as entering this command into your terminal:
```
pip install colorama
```
### OR
```
pip3 install colorama
```
- access to terminal or bash shell requiredcat: n: No such file or directory
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

The previous slide demonstrated the execution of the help file and then the program. This slide depicts the program executing without the help menu printing.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    JUPYTER

chris@DESKTOP-QRQO39E:~/T1A3$ ./run_movie.sh
colorama==0.4.5
Do you want to print the help file? (y/n)
n
1.Show movie list
2.Add a movie to our database
3.Edit the rating of a movie on our database
4.Delete a movie from our database
5.Rent a movie from our collection
6.Exit
Select an option (1-6):
```

You have reached the end of the presentation. Thanks for watching/reading.