CS 5001
STUDENT NAME(S):
SEMESTER:

Application Design Activity: Star Rating App

The goal of this activity is to walk you (or a group of people) through the “big picture” of the program, to the “little picture” of implementing each function making up the program. While this process may seem tedious at first, know you are training your mind to look at these problems, and it will help tremendously when applications get very large.

Scenario

Your client has hired you to update the Star Rating Application. You now have a better understanding of functions and would like to focus on designing the application to make use of functional code. Your hope is by using good design now, you will have minimal updates later.

Product Specifications

Your client wants a star rating application that lets the client enter in both movie and a rating. The application will keep track of both movies and ratings, and upon request will print out all the movies entered along with their associated stars. They have provided you with the sample interface with output.

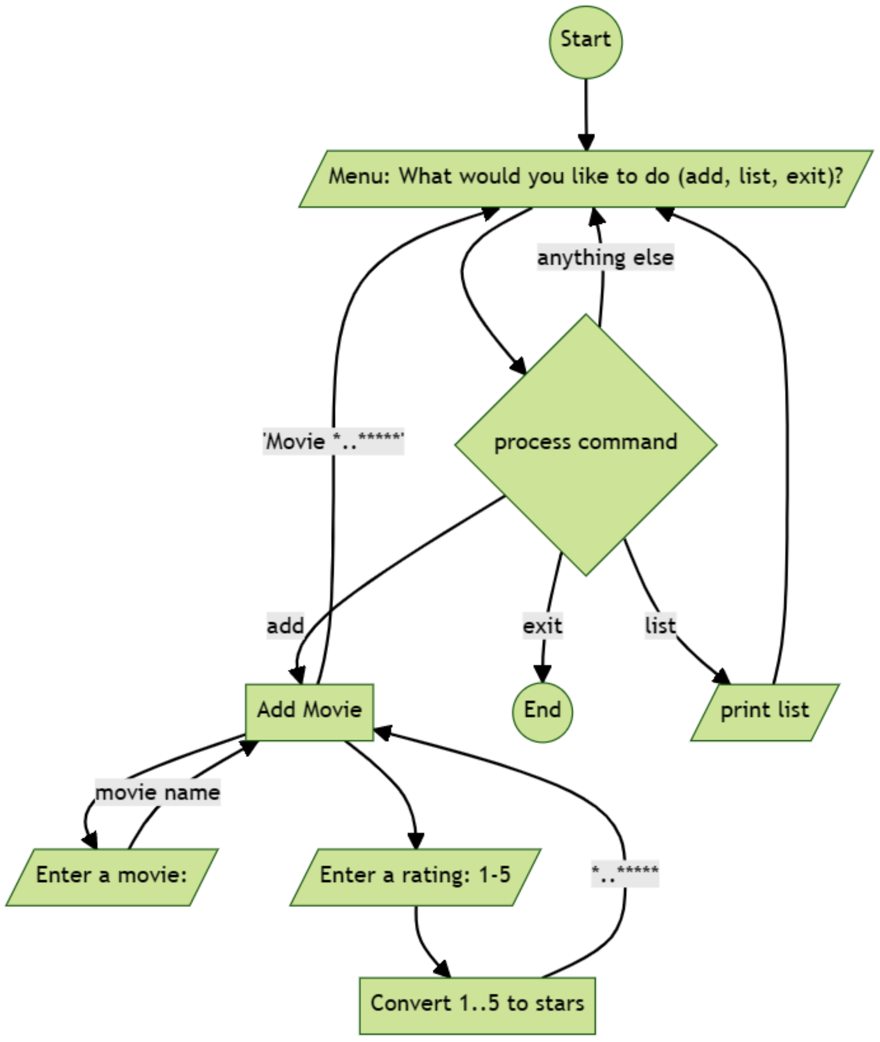
```
What would you like to do (add, list, exit)? add
Enter a movie: V
Enter a rating 1-5: 5
What would you like to do (add, list, exit)? add
Enter a movie: Princess Bride
Enter a rating 1-5: 100
What would you like to do (add, list, exit)? add
Enter a movie: Jurassic Shark
Enter a rating 1-5: 1
What would you like to do (add, list, exit)? Add
Enter a movie: Star Wars Christmas Special
Enter a rating 1-5: 2
What would you like to do (add, list, exit)? list
*****  V
*****  Princess Bride
*      Jurassic Shark
**     Star Wars Christmas Special

What would you like to do (add, list, exit)? exit
```

Your client has added the following constraints

- Add, list, exit – words will be exact, but case shouldn’t matter. As such, they can enter Add, ADD, add – and all will work
- Any other word will just be ignored printing the menu again.
- Like before, anything above the lowest number of stars (1) and the highest (5) will be moved to the lowest/highest.
- You can safely assume the client will enter numbers (no need to error check)
- Partial numbers (1.5) are valid and will be rounded down.
- You don’t need to store the movies separately (you can just store it as a string separated by \n)
    - Additionally, trailing spaces/blank lines will be ignored
- While the stars are padded to be maximum + 2, you are not required to follow that spacing.
    - You are required to start with the stars and have at least two spaces after the star.
    - If you feel like a **challenge** you can look at Python f-string cheat sheet (fstring.help)

Big View: Program Flow

Write a flow diagram for the program in its entirety. While this isn't always possible to do, for smaller programs it is possible and encouraged. This is slightly different from writing a code flow diagram, in that it won't translate directly to code, but it will help you visualize the program.



Decomposition

Using the flow diagram (and your own feeling). Attempt to break the program down into individual components. This is called problem decomposition. At the very least, you are looking at the following the following components

- Add: Add a movie to string of movies separated by "\n"
- List: Prints out the string list of movies
- Menu: displays a menu, asking what to do
    - If exit is typed, the program ends.

Using this page, write out each 'part' in a box. Required in the box are:

- Arguments – what is needed for this part to work
- Requires – what has to be written first for this to work
- Short (a bullet description of the work needed – go back to client requirements)
- Does each part have **one** task? If not, can you break it up more?

Menu:
- Prints menu to screen
- Makes word lowercase
- Returns client choice

Arguments: nothing
Requires: nothing
Called by: Process Command/Run
See: input(), str.casefold()

Run/Process Command
- Calls menu
- If/else on result
- Calls add or list
- Keeps track of movie string

Arguments: none, maybe movie string?
Requires: menu(), add_movie(), list_movies()
Called by: main() or as the main()?

List movies
- Print outs the string containing all the movies added

Arguments: movies
Requires: nothing
Called by: command/run
Yes, this may be one line – but it could be expanded in the future!
See: str.strip() (not required but removes trailing \n from the movies string)

Add Movie:
- Gets a movie and rating
- Formats both
- Returns the Movie + Star Rating to be added to the list

Arguments: nothing
Requires: get_movie from client, gets rating from client, convert rating to stars
Called by: command / run

Get movie
- Prompts client for a movie
- Returns single movie name

Arguments: none
Requires: nothing
Called by: Add movie
See: str.strip() – good habit to clean up client input

Get rating
- Prompts client for movie rating
- Can be a float number (convert str to float)
- Need to be int between MIN and MAX (1-5) stars, if statements
- Convert to int
- Calls convert rating to turn to str
- Returns star string

Arguments: none
Requires: MIN, MAX (consts)
Called by: Add movie

Convert Rating
- Converts an int to a star rating
- Limited by MIN and MAX stars
- Returns a string * to *****

Arguments: int
Requires: nothing
Called by: get_rating

There is no set way to do the blocks, but the important part is ask yourself is each block only performing **one task** by itself. It may call other methods to build on tasks, that is fine – but for the actual work, ideally one task.

These seven blocks are meant to match up with most flow diagrams one block for each major component. They will also translate to each function.

Small View: Writing Functions

For the next part, you stop thinking about the entire program and hyper focus on each function! This means you treat each function as an individual program to be completed with a **small/simple** task.

Given the blocks above, we can create the following functions

- run()
- menu() -> str
- list_movies(str) -> None
- add_movie() -> str
- get_movie() -> str
- get_rating() -> str
- convert_rating(int) -> str

If this format seems familiar, it is because this is very close to a function definition! With all functions, you want to follow these steps

1. define
2. document
3. implement
4. test

This design walk through helps you figure out the define for each function. Your next step is to pick one function. There are a variety of philosophies on where to start. We recommend start with the smallest, most isolated function. Isolated in that the work done can be independent from the rest of the program. Given those conditions, which function do you want to start with?

➢ convert_rating(int)

The following could all work: list_movies, get_movie, and convert_rating. As the first two require client input, we choose convert_rating as it is easier to test.

- Define the function, you can write 'pass' in it for now
- Define the test function in test_star_rating_app.py
- Document the function including **examples**
  - Remember, the 'rest' of the program doesn't exist, so just think about getting an argument of a whole number, and returns stars between MIN and MAX.
- Implement the function
- Using tests, test the function – making sure it works in a variety of conditions

<u>**Repeat this process for each function, until the final program can be tested!**</u>

➢ star_rating_app.py has a template you can use in the HW3 github

Constants?

As you write code, you will find places to "hard code" values such as 1 or 5. It is better to make those constants at the top of the file, so your program can easily modify the "literal" values.

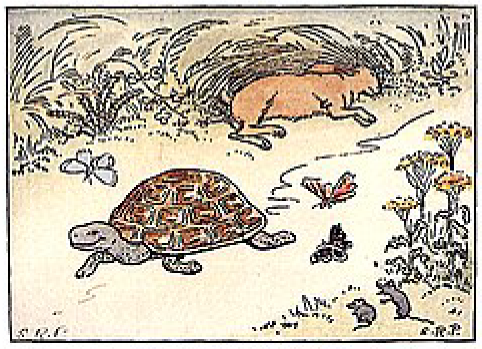
One last IMPORTANT note:

This design process may have seemed like a lot, but as programs grow in size, taking a moment to think and write about your design on paper will go wonders towards developing your program.

Furthermore, testing as you write seems slow, but it speeds up your overall time in the end.

Remember to

- Start early
- Write often
- Take breaks
- The steady Tortoise beats the Hare


THE TORTOISE AND THE HARE