Python Programming

Unit 4 Lab: Carefully Adding a Bit More Logic

Overview

Welcome to the Unit 4 lab!

Our goal is that, at the end of the next lab (you're almost there!), your app makes it possible for users to search for a movie and print out either the Rotten Tomatoes rating or the search results.

Let's get a little closer. We still have hard-coded values, but in the next lab we'll actually get the real ratings. For now, let's incorporate some intermediate variables and error-catching.

Deliverables

You're going to continue building this locally from the last lab. You'll write all of your code in the same `movie_app.py` file.

Run the file from the command line to check your work.

Reminder: On your laptop, you can run the file from your command line with the following:

```
python movie_app.py
```

> **Hint:** Make sure you are printing something out with the `print` statement. Otherwise, you won't see any output from running your program!

Requirements

1. Your `get_movie_rating` takes a source argument and prints a message if the source isn't found.
2. Your `Movie`'s `rating` holds a list of dictionaries.
3. Your `if` block in the `main` function is now in a `while` loop that continues until the user puts in a valid input.

Directions

Augment the code you wrote for the Unit 2 lab.

Part 1: Making the Rating More Advanced

Part 1a: The Rating Format

Right now, `search_or_ratings` being `2` calls `print_single_movie_rating`, which creates a `Movie` object with the rating, which is currently hard-coded to `7`.

However, we don't want a hard-coded rating, right? We want a Rotten Tomatoes rating — we just don't have it yet.

It would be nice to have the hard-coded rating labeled with the source `"hard coded"`. Then, when we add the Rotten Tomatoes rating, we can label that with `"Rotten Tomatoes"`. This way, we can prepare to get the real rating but still have testable code.

Instead of being a single integer, let's have the `Movie rating` be a list of dictionaries. *(Why? So we can store multiple bits of information.)* Our object will look like this:

```
"Ratings":[
  {"Source" : "<what the source is>","Value" : "<numerical value"},
  {"Source" : "<what the source is>","Value" : "<numerical value"},
]
```

For example:

```
"Ratings":[
  {"Source" : "Rotten Tomatoes", "Value" : "54%"},
  {"Source" : "Hard Coded", "Value" : "7"}
]
```

To accomplish this, we need to do two things: 1. Change what we pass in to `Movie` for `ratings` to have a list of dictionaries instead of a single value. 2. Change `get_movie_rating()` in the class to look in the list for the right value, instead of returning a simple value.

Let's tackle No. 1. We are currently only making `Movie` objects in the function `return_single_movie_object()`, which has the code `Movie({'title': movie_title, 'rating': movie_rating})`.

Change the `return_single_movie_object()` function to:

```
def return_single_movie_object(movie_title, movie_rating):
    """
    Take in the movie title and rating, and return the movie object.
    """

    rating_list = [{"Source:" : "Hard Coded", "Value" : movie_rating}]

    return Movie({'title': movie_title, 'rating': rating_list})
```

~~~~~
````

On to No. 2: Go to the `get_movie_rating()` method in your `Movie` class. Let's add
a `for` loop that looks through this list for the source we want and then returns
that value. This way, we can later specify what source we want.
```python
def get_movie_rating(self):
    """
    get_movie_rating is a getter function that returns the rating.
    """

    # Loop through each rating and return it if the source is "Hard Coded".
    for ratings in self.movie_data["rating"]:
        if ratings["Source"] == "Hard Coded":
            return ratings["Value"]
~~~~~

Now let's test it out. Set `search_or_ratings` to `2` and run the program.

We get a `KeyError: "Source"`. What happened? The key `"Source"` isn't in the dictionary. Hmmm. Can you look at the two new functions above and see why?

If you look at the code we just added to `return_single_movie_object()`, you'll see that we have `"Source:"` as the key — not `"Source"`. The smallest typos make huge differences! Delete the `:` and run it again. It works!

Part 1b: Error-Checking

If the goal is to be able to change which rating we want to print, we'll need a way for a user to specify that. We'll change `def get_movie_rating(self)` to take in a `source` argument and give the parameter a default value for anyone who forgets to specify.

```
def get_movie_rating(self, source="Hard Coded"):
    """
    get_movie_rating is a getter function that returns the rating.
    """

    # Loop through each rating and return it if the source is what's passed in.
    for ratings in self.movie_data["rating"]:
      if ratings["Source"] == source:
          return ratings["Value"]
```

Because we've changed something, test it out. Great!

Now, let's say someone calls this method with `source` equal to `"Rotten Tomatoes"`, which we currently don't have. What will our app do?

Let's try it. In `print_single_movie_rating()`, change the `print` statement to `print("The rating for", my_movie.get_movie_title(), "is", my_movie.get_movie_rating("Rotten Tomatoes"))`. Now run it:

```
The rating for Moana is None
```

Hmm. That's probably not true, right? I'm sure *Moana* doesn't have a `None` rating on Rotten Tomatoes, but that's what Python returns because it could find it in our list. It'd be better if `get_movie_rating()` told us that it didn't have that key. Let's add a formatted exception to the bottom of `get_movie_rating()`, which tells Python to throw an error:

```
# If no matching rating is found, we will raise an error.
raise Exception("Rating for source {0} was not found!".format(source))
```

Now run it.

That's a little extreme, right? Let's find a happy medium:

```
def get_movie_rating(self, source="Hard Coded"):
    """
    get_movie_rating is a getter function that returns the rating.
    """

    # Loop through each rating and return it if the source is what's passed in.
    for ratings in self.movie_data["rating"]:
      if ratings["Source"] == source:
          return ratings["Value"]

    # If the code makes it here, it hasn't returned in the for loop.
    return "- Wait - Rating for source {0} was not found!".format(source)
```

Try that. Better, right? Now, before we forget, go remove the `"Rotten Tomatoes"` parameter from the call in `print_single_movie_rating`.

Part 2: Adding a `while` Loop

> When you follow these directions, be careful that you don't create an infinite loop! If your program doesn't stop, you can hit `ctrl-c` in the terminal window to interrupt the program.

Let's look at our `main` function — particularly, the `if` block. Right now, the function ends after the user inputs anything. However, if they input something wrong, instead of printing an error and ending the program, it'd be a lot nicer to print the error and let them try again.

Let's put our `if` block in a `while` loop. Something to note about loops: You can exit them using the keyword `break`.

Change your `if` block in `main` to look like this:

```
# We set up an infinite loop (while True) so that we can keep asking the
# user the same question until they give us valid input ("1" or "2"). As
# soon as a valid input is reached, the appropriate function runs and the
# loop is terminated with "break."
while True:
  if search_or_ratings == 1:
      # If search_or_ratings is 1, call list_search_results().
      list_search_results(default_movie_list)

      ## **NEW LINE HERE** - If the user had a valid choice, leave the while loop
      break

  elif search_or_ratings == 2:
      # If search_or_ratings is 2, call print_movie_rating().
      print_single_movie_rating("Moana")

      ## **NEW LINE HERE** - If the user had a valid choice, leave the while loop
      break

  else:
      # If search_or_ratings is otherwise, give an error.
      print("Error: Your input must be 1 or 2!")

      ## NOTICE NO BREAK HERE - The user didn't input something valid, so let's
give them another try.
```

Compare your code to ours. Remember, if you set `search_or_ratings` to something other than 1–3, `ctrl-c` will stop the loop.

Part 3: String Formatting

Quickly, let's do one last thing. In `print_single_movie_rating`, let's change the `print` statement to have string formatting. You can just use this code:

```
# Print the rating. Note that we have to escape the quotes around the movie
# title because those quotes are inside a string that also uses quotes.
print("The rating for \"{0}\" is {1}.".format(my_movie.get_movie_title(),
my_movie.get_movie_rating()))
```