# Assignment 3: Movie Chatbot Continued

The original version of this assignment is taken from the textbook Concrete Abstractions: An Introduction to Computer Science Using Scheme, by Max Hailperin, Barbara Kaiser, and Karl Knight, Copyright (c) 1998 by the authors. Full text is available for free here. Assignment evolved at Pomona College through several instructors' offerings, with changes by Nathan Shelly and Sara Sood, Northwestern University.

In this assignment, we begin to construct a natural language query system. We will create a program that will participate in dialogs like the following.

```
Welcome to the movie database!

Your query? What movies were made in 1974?
Amarcord
Chinatown

Your query? Who acted in The Dresser?
Albert Finney
Tom Courtenay
Edward Fox
Zena Walker

Your query? What movies were directed by Federico Fellini?
Amarcord

Your query? Bye

So long!
```

There are several parts to the program, and it is convenient (essential, even!) to consider them separately. In Assignment 2, we considered the textual patterns and how we can match them to real user input.

To perform actions on a match, we will write functions that attach to the patterns we recognize. We will associate these functions (actions) to the patterns and complete our movie chat bot system. See the next page for a description of your work.

Notice in the `a3.py` starter code, we have this line: `from match import match`. This brings the match function into our code so that we can use it, just like any other function - `match(["an", "_", "example", "pattern"], ["an", "awesome", "example", "pattern"])`.

## Your Work

1. Implement the action functions. The functions have docstrings and asserts in `a3.py` that demonstrate their intended behavior. This includes the following:
   - `title_by_year`
   - `title_by_year_range` (where the range of years is inclusive, meaning that we also return movies made in the years provided)
   - `title_before_year` (exclusive meaning that we only return titles of movies made BEFORE that year, not in the year)
   - `title_after_year` (also exclusive)
   - `director_by_title`
   - `title_by_director`
   - `actors_by_title`
   - `year_by_title` (*Note* - all the other functions return lists of string(s), this function returns a list of int(s))
   - `title_by_actor`
2. Implement `search_pa_list` (given the provided docstring and asserts)
3. Come up with a new action and write a new pattern/action pair (add to `pa_list`) that utilizes that action. Demonstrate usage of this pattern/action pair by writing a new assert for the `search_pa_list` function that shows the new questions that the user can ask. For example the code below tests the `director_by_title` action. *Note:* `sorted` here is a function that sorts the return list.

```
# two lines here for formatting purposes
# if you wrap an expression in parens () Python allows it to span lines
assert (sorted(search_pa_list(["who", "directed", "jaws"]))
        == sorted(['steven spielberg']), "failed search_pa_list test 2")
```

When grading, we will sort your results as done above to guarantee that we test only the contents of the list, not the order.

Some assert statements are provided in `a3.py` but you should write more asserts (in addition to interacting with your system as a user) to properly test your code.

Complete your work in `a3.py`, and upload it to canvas. You do not need to upload any of the other files provided; we have copies for grading.

**IMPORTANT: Before you submit, please comment out all assert statements. If any of the assert statements in your submission fail, the autograder will not be able to run your code. Additionally, make sure that the query_loop call is commented out.**