Lab 10: Working with multiple classes

Labs are graded for participation rather than correctness. Keep all your lab code in your course GitHub repo to receive credit for your work. We'll be looking to see that you have at least partially completed all problems in each lab.

If you finish the lab assignment early, you may get started on the homework.

Professional skills

If you have not yet participated in collaborative coding, consider collaborating for this lab. We'll set up some breakout rooms for you to self-select to work with other students. See lab 7 for setup instructions. You can also earn professional skills points by volunteering to walk through your lab solution toward the end of class.

The Assignment

For this lab, you'll write multiple classes that interact with each other.

Part 1

Your program will enable a user to search their TV channels for shows based on criteria of their choosing e.g. shows starring a particular actor.

You will need 3 classes: `Actor`, `Show`, and `Channel`. Each class should have the following attributes to start with:

- class `Actor` should have:
  - a first name
  - a last name
  - a collection of shows they appear in
- class `Show` should have:
  - a title
  - a collection of cast members
- class `Channel` should have:
  - a name
  - a number
  - a collection of shows broadcast on the channel

Next, create a file called main.py, import the classes you created, and define the `main()` function.

Create some objects that you can use to test your classes as you build out the `main()` function. You should have at least 3 actors, at least 3 shows, and at least 2 channels. Make sure that some of your actors appear in multiple shows.

Here's an example:

```
actor1 = Actor("Actor", "1")
actor2 = Actor("Actor", "2")
actor3 = Actor("Actor", "3")
show1 = Show("Monday Show", [actor1, actor2])
show2 = Show("Tuesday Show", [actor1, actor2, actor3])
show3 = Show("Friday Show", [actor2, actor3])
channel1 = Channel("DEF", 42, [show1])
channel2 = Channel("XYZ", 31, [show2, show3])
```

In `main()`, create a list variable containing the `Channel` objects that you created. For example, following on from the code above: `channel = [channel1, channel2]`

Searching by actor

Add methods to your classes that will make it possible to get a list of shows on a particular channel that star a particular actor. For example, searching `channel1` for `actor1` would return `[show1]` and searching `channel1` for `actor3` would return an empty list, `[]`.

If you're not sure where to start, this is how our solution works:

- The `Channel` class has a method called `get_shows_by_actor`, which takes an `Actor` object as a parameter and returns a list of `Show` objects. The method iterates through each `Show` object in the `Channel`'s show list. If the `Show`'s cast list contains the `Actor` (determined using the method described in the next bullet point), the `Show` is added to the list to be returned.
- The `Show` class has a method called `cast_contains`, which takes an `Actor` object as a parameter and returns `True` if that `Actor` is in the `Show`'s cast list and `False` otherwise. **Important:** You will find that using syntax like `<object variable> in <collection>` and `==` to compare two objects doesn't work as expected! We will cover a solution to this in lecture 10. For now, you will have to use object attributes to figure out if an object is in a list or if two objects are the same.

Next, add a function to main.py that will iterate through all available channels, searching for shows starring a particular `Actor`.

In our solution, main.py contains a function called `shows_starring` that takes two parameters: an `Actor` (`actor`) and a list of `Channel` objects (`available_channels`). This function iterates through `available_channels`, calling `get_shows_by_actor` on each `Channel`. The lists obtained from each `Channel` are then concatenated and returned. Remember that you can concatenate lists using `+` or `+=`. For example, `list_a += list_b` adds the contents of `list_b` to `list_a`.

Informally test your function (and the methods it calls) in `main()` by printing the list of shows starring a particular actor. **Important:** You will find that printing instances of your own classes doesn't work as expected. We will cover a solution to this in lecture 10. For now, instead of printing individual objects, print the values of their attributes e.g. if `show` is a `Show` object, use `print(show.title)` instead of `print(show)`.

Part 2

Extend the program you wrote in Part 1 by adding new functionality. The purpose of this part of the lab is to think about program design: what attributes and methods should be added to which classes and what should be handled in main.py. There is no single right answer!

You can assume that the new functionality would be called from `main()`.

Implement the following or come up with your own features:

- Get channel information for a particular show.
- Add a daily schedule to each channel that stores which shows are playing on a particular day. In the real world, you'd want to have show time as well as day but we'd suggest keeping it simple for now.
- Get all shows playing on a particular day across all channels.