University of Hull

Department of Computer Science

Wrestling with Python – Week 04
Using Lists

Vsn. 1.0 Rob Miles 2013

Before you go onto this lab, please make sure that you have sorted out the Cinema Entry program from last week. This lab builds upon the program (although in this lab you will effectively be starting a new program to add new functionality)

Using Lists

In this laboratory we are going to explore how to use an array to store data, and how we can avoid having to "hard wire" items into programs.

```
Welcome to our Multiplex
We are presently showing:
    1. Rush (15)
    2. How I Live Now (15)
    3. Thor: The Dark World (12A)
    4. Filth (18)
    5. Planes (U)
Enter the number of the film you wish to see: 1
Enter your age: 12
Access denied – you are too young
```

Above you can see the menu from the cinema entry program. At the moment we are creating the menu by writing out the individual film items:

```
print ("    1. Rush (15)");
print ("    2. How I Live Now (15)");
print ("    3. Thor: The Dark World (12A)");
print ("    4. Filth (18)");
print ("    5. Planes (U)");
```

This will work, but it means that every time the films change we have to make a new program. What the program should do is read in the film names at the start, and then display them for each customer. Later we can make the program read the names from a file, or perhaps even a web feed.

Creating a New Python Script

1. Log in and start up Idle.
2. Use **File>New Window** to open a new window.
3. Use **File>Save As** to save the file. Give it a new name, like **cinemaList.py**. Don't forget the .py extension so that Idle can tell what type of file it is loading.

Adding a film list

We are going to use a list of strings to hold the names of the films:

```
filmNames = []
```

This statement creates an empty list.

Reading in the film names

Now that we have a list we can read in 5 film names. The best way to do this is to use a loop. In python, the block of code inside a for loop is executed for each value in an **iterable** type. Iterable types are types that can be iterated over, such as ranges, lists and strings.

```
# Read in the film names
for i in range(5):
    filmName = input ( "Please enter the film number " + str(i + 1) + ":" )
    filmNames.append(filmName)
```

This loop will read in the names. In this case, variable **i** is assigned the values in range one by one, and for each value assigned to **i** the block of code inside the loop is executed. In this case the values in range are **[0, 1, 2, 3, 4]**. You can now access each value in the list of stings using a zero based index inside square brackets, like this:

```
filmNames[0]
```

We know that the list is indexed from 0 (in other words the first element in the array has the subscript of zero. However, users would not expect to be counting from 0 in situations like this. Can you see how the programmer has dealt with this?

4. Add the declaration of the list and the loop that reads in the film names. Run the program. Of course, at the moment it will not print anything because we are just reading in the names.

Displaying the Menu

Once you have the array set up in the program you can use it to display the menu from which the user will select the film that they want to see. You can use exactly the same for-loop structure before (but you have to make a **new** for loop to display the menu).

5. Add a for loop to print out the names that were entered – accessing each name using filmNames[i]. Remember to add the index plus one to the beginning of each film name string as a means for the user to choose a film.

   Aside: As a list is iterable you can also use a list directly in the loop declaration like this:

   ```
   for name in filmNames:
   ```

   Then variable name will iterate over each of the film names in the filmNames list. However if you use this method you may need to do a little extra work to keep track of the position in the list.

Finding the selected film

The user will enter the number of the film that they want to see. However, they will start counting at 1, whereas the film at the start of your array has the subscript 0. This means that you will have to subtract 1 from the number they enter, to find the name of the film that they have selected.

Handing the film age ratings

This seems to work well, but at the moment it looks like we have no way of handling the age ratings of the films. Since that is what the program was created for, we need to sort this out.

It turns out that the Python string type has a really useful method called `endswith`. You can use it to see if a string ends with a particular piece of text. This is just what we want, since the age rating is given right at the end of the film name. We will use a construction like this:

```
if filmNames[chosenFilm].endswith( "(12A)" ):
    # The user has selected a film that is 12A rated
```

In the code snippet above the variable `chosenFilm` holds the number of the film that was selected by the customer. This will be an integer in the range 0 to 4 (remember that arrays are indexed from 0). The number was obtained by asking the user for the number of the film they want to see, and then subtracting 1 from it.

The if condition looks for the string `(12A)` on the end of the name, so that it can detect films of that rating.

6. Use the above technique to allow the correct film age ratings to be selected. Your program can use the age rating information on the end of each film name rather than having the ratings for each film "baked in" to the program itself. You will need to add code produced in

previous labs to create a fully working solution.

Tidying Up

You can use the above techniques to make the ultimate Cinema Entry program. Here are some thoughts to make it even better.

Use the Length property of the array to control your loops

At the moment you are counting up to 5 each time you work through the list. It turns out that there is a method to find the length of a list. It tells the program how many elements the list contains:

```
for i in range(len(filmNames)):
```

This means that if the size of the list changes (for example if more films are shown one week) the program will still work correctly. Note that this will not work at the start of your program before the list has been populated with film names as initially the size of the list is 0.

Allow the user to enter how many films are being shown

You could ask the user how many films are being shown and then make loop where the user enters the film titles loop that many times. This would also make your program quicker to test, because you could enter the value 1 and just enter a single film name.

Add data tidying error checking

The technique of using `endswith` to get the age ratings will work, but if the user doesn't type the rating value correctly, or adds a space on the end of the film name by mistake the rating behaviour will not work as it should.

There is a method called `strip` that you can use obtain a version of a string with all the leading and trailing spaces removed:

```
trimmedName = enteredName.strip()
```

The above statement sets the variable `trimmedName` the value of `enteredName` (which must be a string) with all the leading and trailing spaces removed.

You should also make sure that when the user enters the film names you check to make sure that each has a sensible age rating. You will have to add some tests to do this. One way would be to compare what has been entered with a list of acceptable rating values. This should be done before a new film is added to the list.

```
filmRatings = ["(U)", "(12)", "(12A)", "(15)", "(18)"]
validRating = False
for rating in filmRatings:
    if(newFilmName.endswith(rating)):
        validRating = True
        break
```

After this code is executed if validRating is True you can go ahead and add the film to the list of filmNames. However, if validRating is False then the new film should not be added to the list, and the user should be prompted to add the information again.

Another more user friendly way would be to ask the user to choose from a set of ratings, and add the rating to the film name programmatically, ensuring that it is always in the expected format.

Rob Miles
October 2013