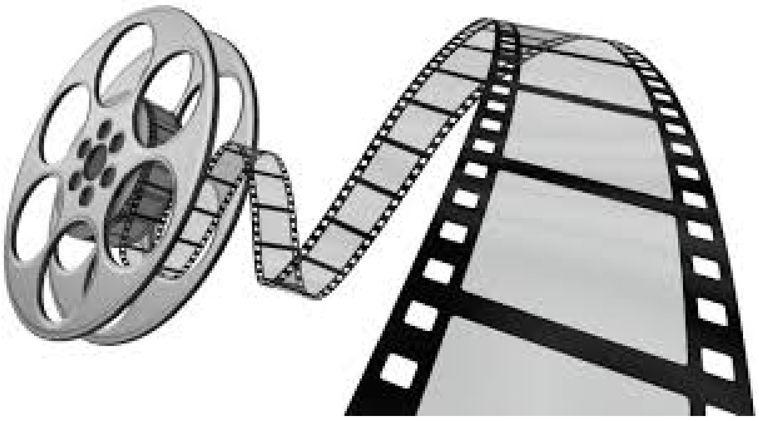
Take-Home Task 2: Popularity Clouds

(Weight: 30%. Due 8 Friday November 2019)

Overview

This task requires you to use several of the technologies introduced in this unit (Python), database queries (SQLite) and web development (HTML).



Goal

This task involves writing a Python program which interrogates an SQLite database and generates HTML documents as its output. It is thus an excellent example of the main theme of this unit, i.e., the way different "computer languages" can be used together to build new IT systems.

The aim of this task is to develop a program which creates HTML documents that visualise movie actors' popularity amongst their fans, according to a survey of movie customers stored in a database. More specifically, given a particular category of movie fans, for example, females or customers whose age is between 30 to 40, we want to know how popular the actors stored in the movie database are with this group. The required output is a set of HTML pages, one per customer group. Each page must contain a visual representation of the actors' popularity in which the more popular actors names are displayed in a bigger font.

The Database

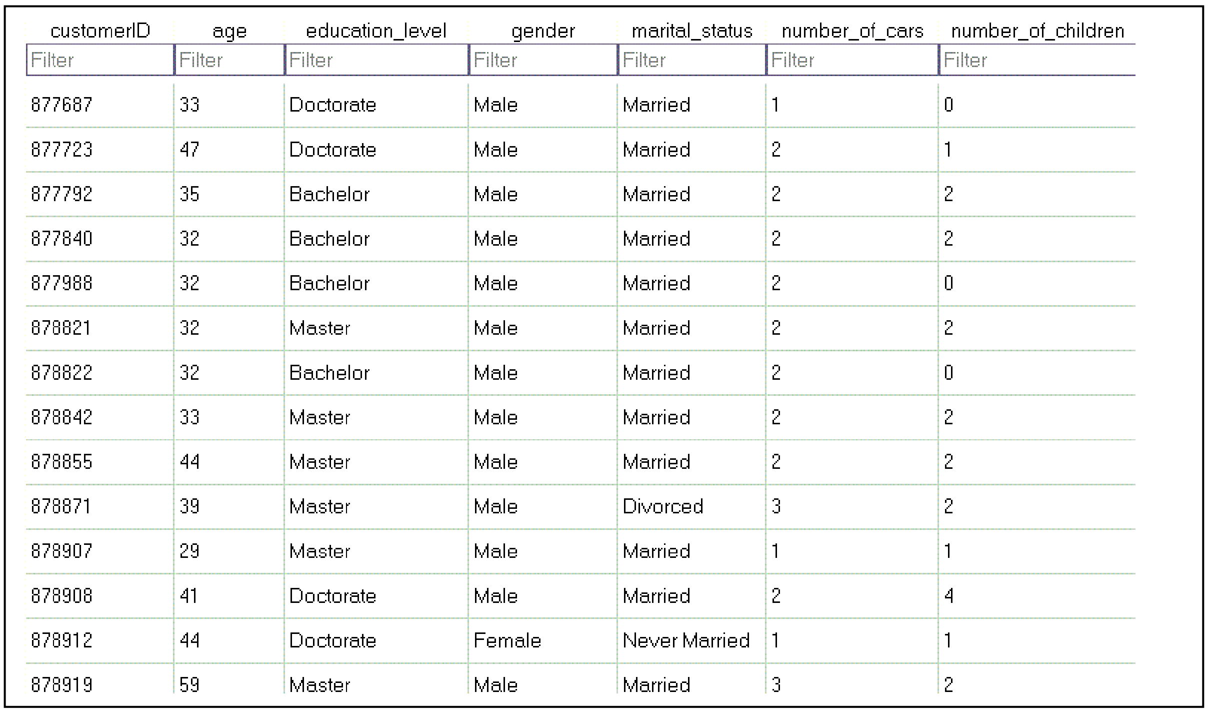
An SQLite database backup file or 'dump' called `movie_survey.sql` containing the data to use for this task has been supplied. It contains results from a survey taken by Microsoft employees about their favourite actors and movies. In order to complete this task, you need to first recreate this database by importing the supplied script in the *SQLite DB Browser*.

There are four tables in this database: `actors`, `customers`, `actors_movies` and `favorite_actors`. Among them, two tables are needed for this task, `customers` and `favorite_actors`. The definition of these two tables is explained below.

Table `customers`

Table `customers` provides information about movie patrons or customers. Each row in the table consists of seven fields as illustrated below. Field `customerID` is the identifier of a customer and is the primary key. Fields two to seven provide the customer's personal information including age, education level, gender, marital status, number of children and number of cars. The fields are defined by the schema below:

```
customerID          INT(6),
age                 INT(11),
education_level     VARCHAR(30),
gender              VARCHAR(10),
marital_status      VARCHAR(20,
number_of_cars      INT(11),
number_of_children  INT(11),
PRIMARY KEY (customerID)
```

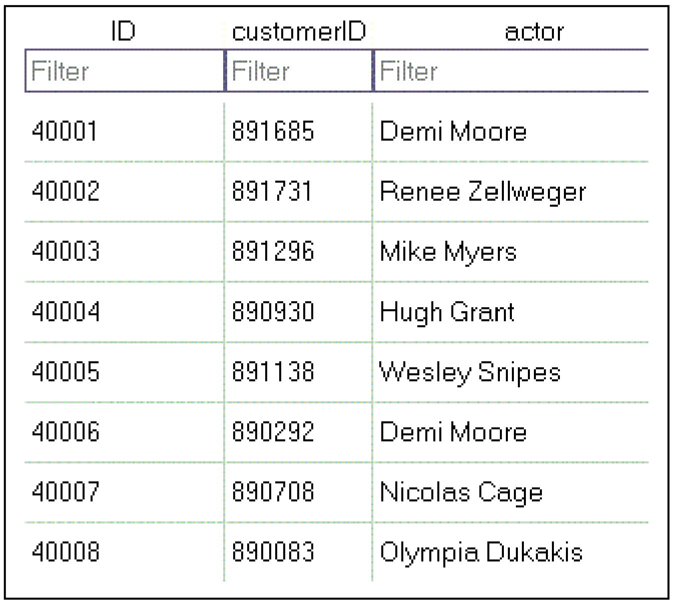


customerID	age	education_level	gender	marital_status	number_of_cars	number_of_children
Filter	Filter	Filter	Filter	Filter	Filter	Filter
877687	33	Doctorate	Male	Married	1	0
877723	47	Doctorate	Male	Married	2	1
877792	35	Bachelor	Male	Married	2	2
877840	32	Bachelor	Male	Married	2	2
877988	32	Bachelor	Male	Married	2	0
878821	32	Master	Male	Married	2	2
878822	32	Bachelor	Male	Married	2	0
878842	33	Master	Male	Married	2	2
878855	44	Master	Male	Married	2	2
878871	39	Master	Male	Divorced	3	2
878907	29	Master	Male	Married	1	1
878908	41	Doctorate	Male	Married	2	4
878912	44	Doctorate	Female	Never Married	1	1
878919	59	Master	Male	Married	3	2

Table `favorite_actors`

Table `favorite_actors` provides information about movie fans' favourite actors. Each row in the table consists of three fields as illustrated below. `ID` is the identifier of this record, `customerID` is the identifier of a customer, and `actor` is the name of an actor who is one of this fan's favourites. The fields are defined by the schema below:

```
ID              INT(6),
customerID      INT(6),
actor           VARCHAR(100),
PRIMARY KEY (ID)
```



ID	customerID	actor
Filter	Filter	Filter
40001	891685	Demi Moore
40002	891731	Renee Zellweger
40003	891296	Mike Myers
40004	890930	Hugh Grant
40005	891138	Wesley Snipes
40006	890292	Demi Moore
40007	890708	Nicolas Cage
40008	890083	Olympia Dukakis

Notice that the format of actor names in this database is "given_name surname" as shown in the tables above. When you order and display the names as required in this task, you can treat each name "given_name surname" as one string without separately dealing with given names and surnames. (This means that the names will be sorted on the actors' *given* names.)

General Requirements

The aim of this task is to visually show the popularity of actors among specific categories of movie fans. You are required to develop a Python program that accesses the SQL database described above to generate HTML documents. The user of your program specifies:

- How many results to display; and
- Which categories of movie fans' opinions to display.

The result is a collection of hyperlinked HTML documents, one per category of movie customer. The different categories of customer that the user can specify are:

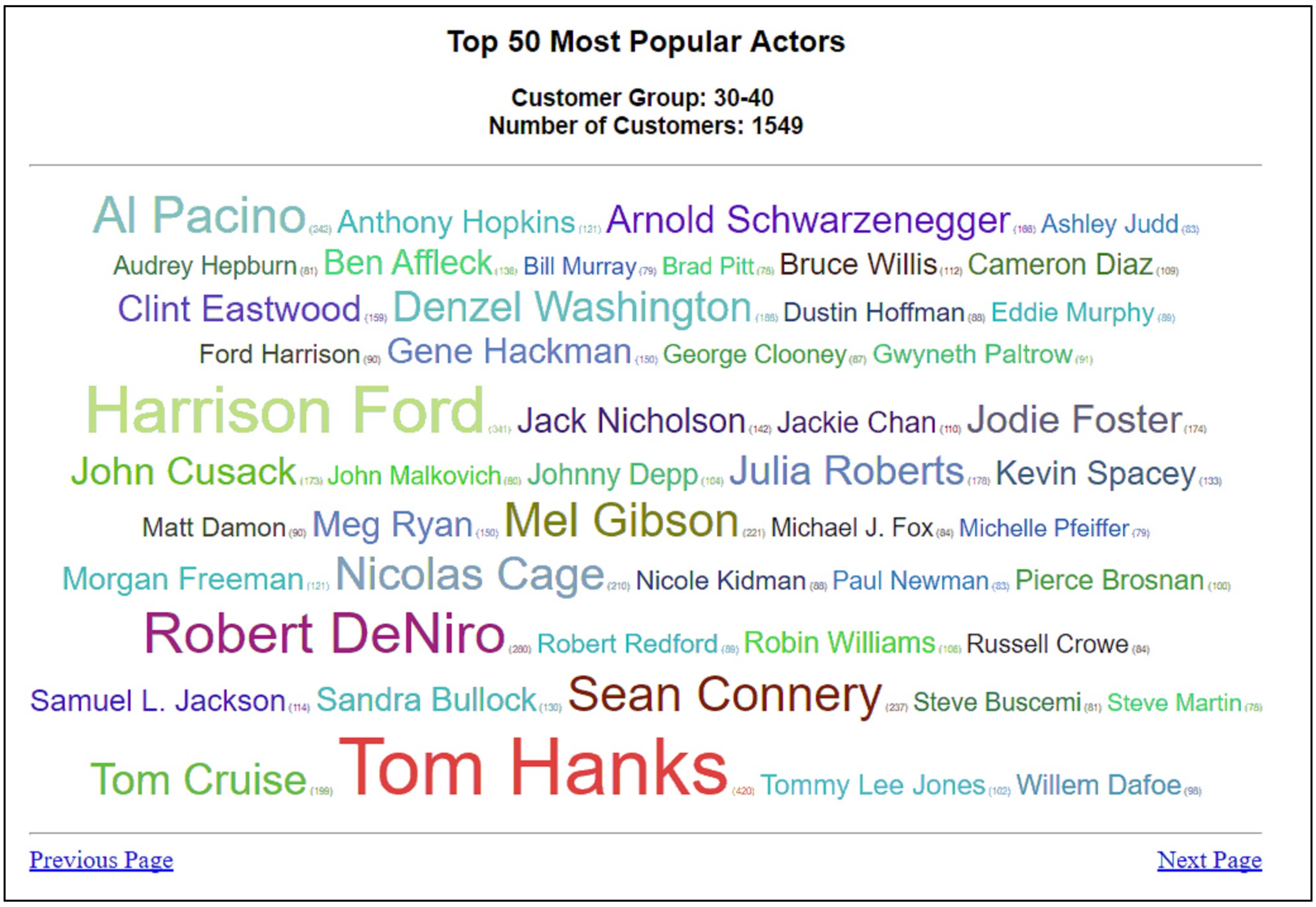
- Male fans;
- Female fans;
- Fans in a certain age range, e.g., 30 to 40, inclusive; and
- All fans.

An example of one of the HTML documents is shown in Figure 1. It shows the popularity of actors with movie customers with an age between 30 and 40, inclusive. (This page is one of those to be produced by Test 2 in the provided template file `popularity.py`.) The popularity of an actor is defined by the number of customers who like this actor.

The idea is that more popular actors are displayed in larger fonts. For instance, *Tom Hanks* is more popular than *Sandra Bullock* because *Tom Hanks* has 420 fans in this age range, while

Sandra Bullock has only 130. In Figure 1, the font used for *Tom Hanks* is obviously larger than that of *Sandra Bullock*. Each actor's name is also followed by the number of movie customers who like this actor, in a smaller, unintrusive font. For instance, in Figure 1, *Tom Hanks* is associated with number 420 because he is a favourite actor to 420 customers.

In this task, we also use different colours to display actors whose popularities are different. Actors with the same popularity are displayed in the same font size and also the same colour. For example, *Dustin Hoffman* and *Nicole Kidman* in the example below have the same font size and colour since both of them are popular with 88 fans. Actors with different popularities are shown in a different font size and also a different colour, e.g., *Tom Hanks* and *Sandra Bullock* in the example below have a different font size and also a different colour. (A possible refinement of the task is to relate the colours, so that more popular actors get a stronger colour, but you are not required to do so.)



Top 50 Most Popular Actors

Customer Group: 30-40
Number of Customers: 1549

Al Pacino (242) Anthony Hopkins (121) Arnold Schwarzenegger (168) Ashley Judd (83) Audrey Hepburn (81) Ben Affleck (138) Bill Murray (79) Brad Pitt (78) Bruce Willis (112) Cameron Diaz (109) Clint Eastwood (159) Denzel Washington (188) Dustin Hoffman (88) Eddie Murphy (89) Ford Harrison (90) Gene Hackman (150) George Clooney (87) Gwyneth Paltrow (91) Harrison Ford (341) Jack Nicholson (142) Jackie Chan (110) Jodie Foster (174) John Cusack (173) John Malkovich (80) Johnny Depp (104) Julia Roberts (178) Kevin Spacey (133) Matt Damon (90) Meg Ryan (150) Mel Gibson (221) Michael J. Fox (84) Michelle Pfeiffer (79) Morgan Freeman (121) Nicolas Cage (210) Nicole Kidman (88) Paul Newman (83) Pierce Brosnan (100) Robert DeNiro (280) Robert Redford (89) Robin Williams (106) Russell Crowe (84) Samuel L. Jackson (114) Sandra Bullock (130) Sean Connery (237) Steve Buscemi (81) Steve Martin (78) Tom Cruise (199) Tom Hanks (420) Tommy Lee Jones (102) Willem Dafoe (98)

Previous Page Next Page

Figure 1: Actors popularity (top 50) with fans aged between 30 and 40 (Test_02, p2)

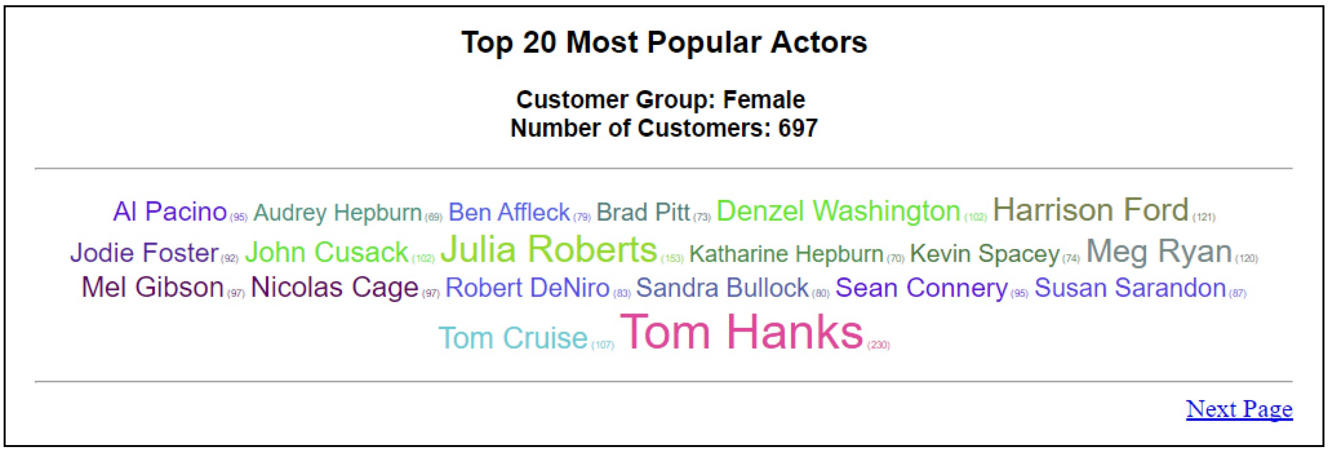
Your job is to develop a Python program which generates such pages using the data in the SQL database `movie_survey`. As illustrated in the Python template file accompanying these instructions, you must define a function called `show_popularity` which, given a list of customer categories, produces HTML files that can be displayed in a web browser, one per customer category.

Another feature of the document shown in Figure 1 is that on the top of the page you must display the category of the customers, e.g., aged “30-40” in this case, and the number of customers in this group, e.g., 1549 above.

In addition, your program must hyperlink each of the pages together, to make it easy for the user to search through them. In Figure 1 above we can see hyperlinks at the bottom of the page to the previous and next categories in the list. However, no ‘previous’ link is produced for the first category in a user’s list and no ‘next’ link is produced for the last page in a list.

For example, if the given list of customer groups is [‘Female’, ‘Male’, ‘30-40’] then your function must generate three HTML pages, one for female customers, one for male customers, and one for customers aged between 30 and 40, inclusive. In the male movie customers’ page, the ‘previous’ link and the ‘next’ link should take us to the female customers’ page and the page for customers aged between 30 and 40, respectively. However, for the female customers’ page, there is no ‘previous’ link, and for the page of customers aged between 30 and 40 there is no ‘next’ link.

Figures 2 to 4 below show the expected output when we ask for the top 20 actors preferred by fans in the category list [‘Female’, ‘Male’, ‘30-40’]. (These three pages are the output expected for Test 1 in the template file `popularity.py`.) In each case the relative popularity of actors in each of the specific customer categories is shown clearly, and each of the three pages is hyperlinked together in the order of the list of customer categories.

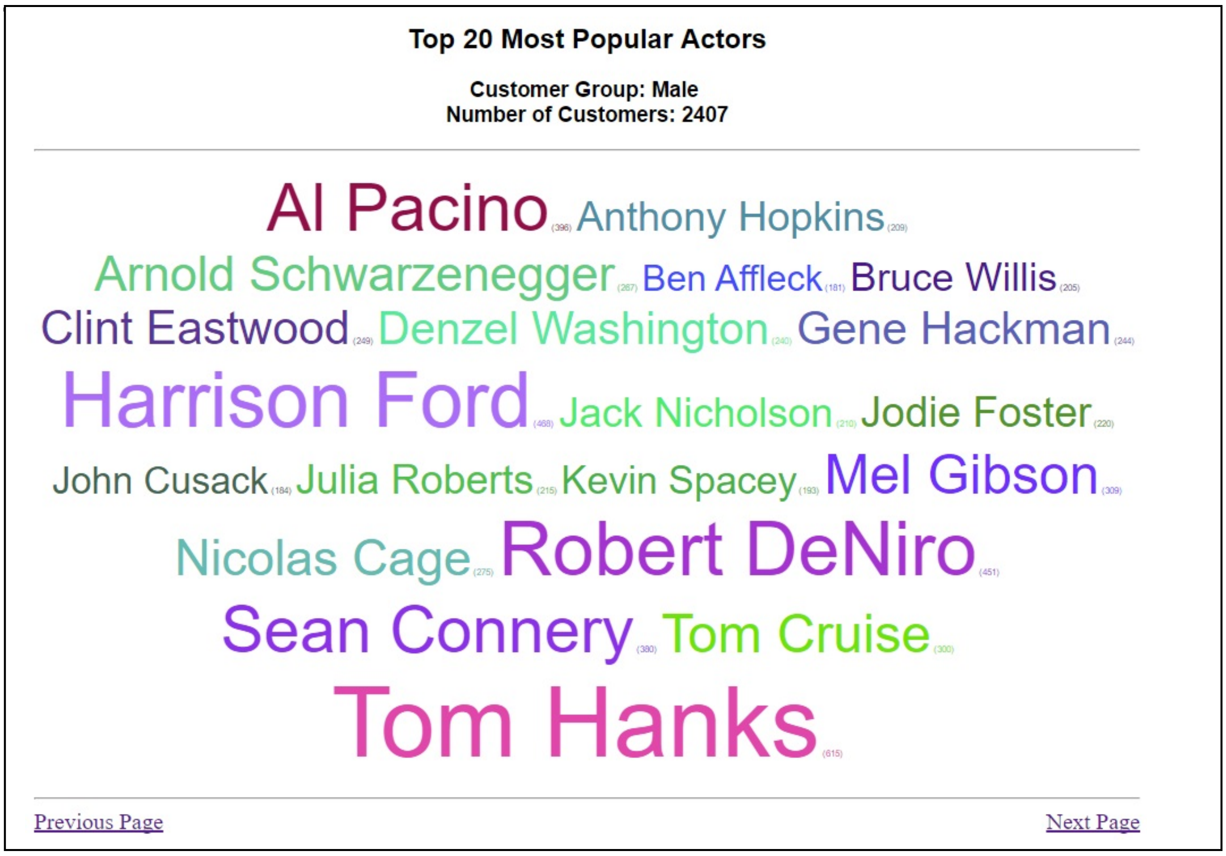

Top 20 Most Popular Actors

Customer Group: Female
Number of Customers: 697

Al Pacino (95) Audrey Hepburn (69) Ben Affleck (79) Brad Pitt (73) Denzel Washington (102) Harrison Ford (121) Jodie Foster (92) John Cusack (102) Julia Roberts (153) Katharine Hepburn (70) Kevin Spacey (74) Meg Ryan (120) Mel Gibson (97) Nicolas Cage (97) Robert DeNiro (83) Sandra Bullock (80) Sean Connery (95) Susan Sarandon (87) Tom Cruise (107) Tom Hanks (230)

Next Page

Figure 2: Actors popularity (top 20) for female fans (Test_01, p1)



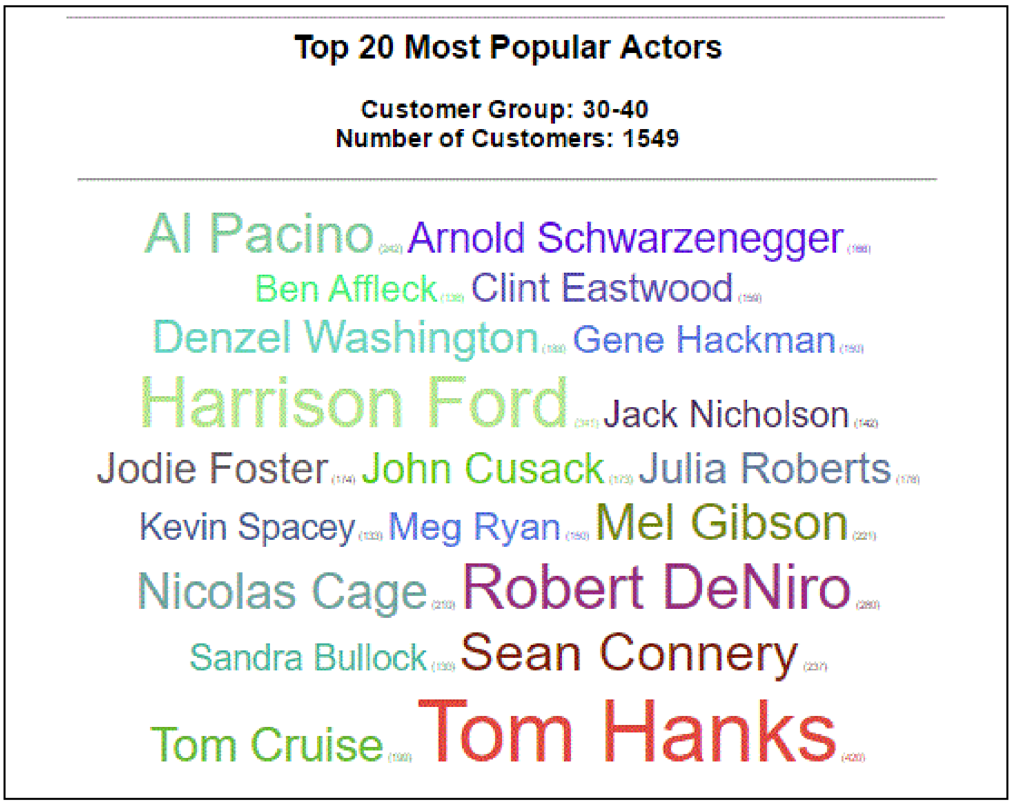
Top 20 Most Popular Actors

Customer Group: Male
Number of Customers: 2407

Al Pacino (396) Anthony Hopkins (209)
Arnold Schwarzenegger (287) Ben Affleck (181) Bruce Willis (205)
Clint Eastwood (249) Denzel Washington (240) Gene Hackman (244)
Harrison Ford (468) Jack Nicholson (210) Jodie Foster (220)
John Cusack (184) Julia Roberts (215) Kevin Spacey (193) Mel Gibson (309)
Nicolas Cage (275) Robert DeNiro (451)
Sean Connery (380) Tom Cruise (300)
Tom Hanks (615)

Previous Page Next Page

Figure 3: Actors popularity (top 20) for male fans (Test_01, p2)



Top 20 Most Popular Actors

Customer Group: 30-40
Number of Customers: 1549

Al Pacino (242) Arnold Schwarzenegger (168)
Ben Affleck (138) Clint Eastwood (159)
Denzel Washington (188) Gene Hackman (150)
Harrison Ford (341) Jack Nicholson (142)
Jodie Foster (174) John Cusack (173) Julia Roberts (178)
Kevin Spacey (133) Meg Ryan (150) Mel Gibson (221)
Nicolas Cage (213) Robert DeNiro (280)
Sandra Bullock (130) Sean Connery (237)
Tom Cruise (190) Tom Hanks (420)

Figure 4: Actors popularity (top 20) for customers aged between 30 and 40 (Test_01, p3)

Specific Tasks

The specific things you must do to complete this portfolio task are as follows. NB: This is a large task. You should modularise your solution into several functions to make your program code manageable and readable. You <u>**must**</u> use the supplied template file `popularity.py` as a starting point.

1. Restore the `movie_survey` database using the supplied dump file `movie_survey.sql`
2. Write a Python function named `show_popularity` that has three parameters: (a) a list of strings representing a list of customer groups, (b) an integer value representing how many actors are to be displayed on the page, and (c) a string which gives a name to this particular multi-page document.

   For example, consider the following function call:

```
show_popularity(['20-40', '40-80', 'All'], 30, 'Test03')
```

The first argument (a list) represents the customer categories required and this example indicates that three pages are to be created: one for customers aged 20 to 40, inclusive, one for customers aged 40 to 80 inclusive, and one for all customers. (Other valid customer categories are 'Female' and 'Male'.)

The second parameter is an integer value which specifies how many actors' names are to be displayed on each page. Those displayed must be the most popular actors for that particular customer group. (Hint: you will want the results set returned from the database query to be ordered by popularity.)

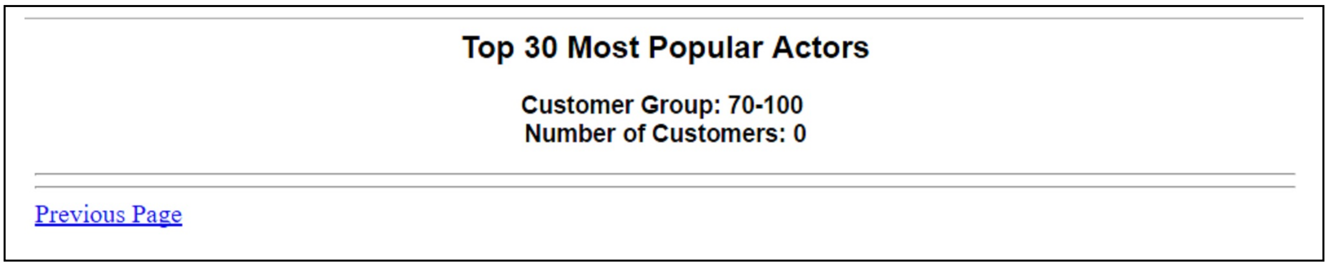
The third parameter is simply a name used to uniquely identify this collection of HTML pages. You should use it together with the customer category to create the name for the HTML files generated. In this particular example your program must produce the following three HTML files:

```
Test03_20-40.html
Test03_40-80.html
Test03_All.html
```

Actors must be displayed in **alphabetical order** regardless of their popularity. As mentioned before, each actor's name in the database consists of a given name and a surname in the format of "given_name surname", so just sort them as a single string. For example, in Figure 1, Al Pacino, Anthony Hopkins, and Arnold Schwarzenegger are displayed in that order, even though Arnold Schwarzenegger has a higher popularity than Anthony Hopkins. Each actor's name must be followed with the actual number of fans derived from the database. Ensure that the numbers produced by your database query match those in the figures above.

At the bottom of each page, you must have hyperlinks to the previous page and next page in the customer category list, if any.

3. Some queries will produce no results at all. In this case you should still produce an HTML page, even if there are no actors listed. An example is shown in Figure 6 below (which is one of the pages produced by Test 10 in the provided template file).


Top 30 Most Popular Actors

Customer Group: 70-100
Number of Customers: 0

Previous Page

Figure 5: A page with no results returned (Test_10, p2)

4. As noted above, the valid customer categories are 'Female', 'Male', 'All' and an age range e.g. '30-50'. If any other category is found in the list provided as the first parameter to `show_popularity`, a message should be printed noting the error and this item should then be ignored. Pages must still be produced for all *valid* cases in the list, however. See the unit tests in the provided template file for examples.

Development hints

- Before you can begin this task you must ensure that you have access to SQLite DB Browser so that you can create the database. You must also have access to the necessary SQLite-Python module so that you can call SQLite functions from Python code.
- Given a movie customer category you will need to execute two queries on the database, one to find out how many customers there are in this category and one to get the list of actors and their popularity. The first query is relatively easy and involves the customers table only. The second involves a join of both the `customers` and `favorite_actors` tables. It is recommended that you develop and debug your database queries in the *DB Browser* before attempting to incorporate them into your Python program. Save your queries to `.sql` files so that you can refer back to them later if you need to.

- Use the screenshots shown above as a guide, but you do not need to duplicate the precise choices of font sizes and colours. Importantly, however, your SQL queries **must** return exactly the same number of fans as shown in the examples above.
- This is a large program, so it is suggested that you tackle it incrementally. Before you even begin coding you should use the *DB Browser* to determine exactly what query your Python program needs to execute. Then develop code to generate just one HTML page. Finally, write the code to iterate over a list of customer groups to generate all documents.
- You need to work out how to define the font size and colour for the text to be displayed on web pages. This can be done by changing the font's 'style'. Refer to the lecture demonstrations and workshop exercises for how to do this.

Deliverables

You should develop your solution by completing and submitting the provided template file, `popularity.py` as follows.

1. This file must include a function `show_popularity` which has all the characteristics defined above. When executed in the presence of an appropriately configured SQLite database, this program must produce the required HTML files.

2. Complete the "statement of authorship" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. **I will assume that submissions without a completed statement are not your own work and they will not be marked.**

You are <u>**not**</u> to submit any SQL database scripts or dumps. The supplied SQL database, and possibly updates of this database, will be used to test your software. Your program must therefore not be hard-coded in any way, as it will be required work with any similarly structured database.

Requirements and marking guide

You are required to extend the provided template by completing the `show_popularity` function (and adding other functions as necessary) so that it can generate the required HTML pages from data stored in the database. Your code must work for all the supplied datasets in the template file **and any other data set in the same format**.

Your solution must pass all the supplied test cases in the template file, and work with any other database of the same format. In particular, your solution must have the following features:

- **SQL queries (4%)**. The SQL queries must be well formed, and produce accurate results from the database for any given category value.

- **HTML pages (2%)**. Produces the required number of HTML pages according to the supplied first parameter (ie the list of categories).
- **Top Actors (2%)**. Each HTML page includes only the number of most popular actors, according to the value supplied in the second parameter.
- **HTML contents (10%)**. Each HTML page is to include:
  - an appropriate title (in the browser tab)
  - a heading including the number of actors listed
  - a sub-heading including:
    - the type of customer group; and
    - the number of customers in that group.
  - hyperlinks to previous and next pages (if any)
- **HTML body (10%)**. the body of the HTML page must display the most popular actors in alphabetic order, with:
  - the size of the actor's name reflecting the popularity (ie more popular actors displayed in a bigger font);
  - actors with the same popularity displayed in the same sized font;
  - a different font colour for difficult popularities; and
  - actors with the same poparlity displayed in the same font colour;
  - the popularity count of each actor, following that actor's name.
  - the fonts and colours used in the HTML must be readable.
- **Python code quality and presentation (2%).** Your program code must be presented in a professional manner. See the coding guidelines in the *JIT104 Code Marking Guide* for suggestions on how to achieve this. All Python and HTML code must be easy to read and understand, thanks to:
  - clear uncluttered layout;
  - concise English comments, explaining each significant code segment's purpose;
  - variable names that clarify their roles;
  - avoidance of 'magic numnbers';
  - code that avoides unnecessary duplication of code;
  - appropriate use of functions to modularise the code;
  - correct grammar and spelling.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive **partial marks for incomplete solutions**.

Academic Integrity

This assignment is for individual assessment only. That means the work you submit must be your own work and not the work of anyone else. You are not permitted to collaborate with your peers on this assignment, other than to discuss high-level strategies. You are not permitted to ask or commission someone else to write the assignment for you, or help you write the assignment.

If you are in any doubt as to what is permitted and what is considered a breach of academic integrity, please talk to one of the teaching staff as soon as possible.

Author:	Colin Fidge (QUT)
Revised:	Donna Teague (QUTIC 2019)