UWI

INFO2602-Assignment 2

JavaScript and Python Foundation

Table of Contents

Introduction

This assignment will be a group-based assignment. Groups consist between 1 and 3 persons. You should carefully consider how to allocate responsibility to finish within the deadline.

The assignment aims to provide students with practice of developing modular components of functionality for building more complex systems as a best practice.

While the assignment is delivered as a group activity for providing opportunity to evaluate your group skills and make the amount of work more manageable, it is important for each member of the group to be able to successfully complete all questions in the assignment.

Ensure that you are on the slack group for the course to share clarification questions. While the tutorial session and classes are useful forms, often the time available creates a significant limitation of how much can be addressed and how much persons benefit. Therefore we encourage you to use the more general forums for clarifications.

Section 1 – The Basics

Write **both** *JavaScript (util.js)* and *Python* Code *(util.py)* for the following questions. We recommend that you use another file (testUtil.js and testUtil.py) to test the functionality of the code written. However, only the util files will be evaluated using our testing suite.

Questions 1 to 4 are based on Figure 1

```
[{
    "name": "John",
    "sex": "m",
    "country": "tt",
    "products": [{
        "name": "Peanut",
        "category": "food"
        "value": 15,
        "date": "2019-03-12"
    }, { …. }, … ]
}, … ]
```

Figure 1 – Structure of a record

1. Write a function called "**getMostValuable**" that accept an array/list of customer records stored as object-literals/dictionaries. Each record has a field called 'products' which stores additional records that contain the fields 'name' and 'value'. This structure is illustrated in Figure 1. The function "**getMostValuable**" will return the position of the customer record with the highest total value from the products they purchased. [5]

2. Write a function called "**addCustomer**" that will add a new customer record to a sorted array/list of customer records. The structure of the customer record is provided in Figure 1. The records are sorted in ascending order based on the name field of the customer record. [5]

3. Write a function called "**getBySex**" that accepts an array/list of customer records stored as object-literals/dictionaries illustrated in Figure 1. The function will return an array/list that contains customers that matches the sex specified as a parameter. The function will accept both the word ['male', 'female'] or the letter ['m', 'f'] as valid inputs. [5]

4. Write a function called "**getCustomerWhoBought**" that accepts an array/list of customer records stored as object-literals/dictionaries and a second parameter which stores the product category. The function will return an array/list that contains customers who has products belonging to the category specified as a parameter. [5]

5. Write a function called "**convertDate**" that will accept an array/list of string dates in the form 'YYYY-MM-DD' and returns an array/list containing valid dates from the values specified in the input array. If an invalid formatted date is found in the array/list, conversion should produce undefined/None for that record. [5]

6. Write a function called "**convertProductDates**" that accepts an array/list of customer records. The function will use the "*convertDate*" function created in Q5 to convert the dates of all the products stored in the records provided as a parameter. [5]

Section 2 – OOP

Write **both** *JavaScript (movieGenre.js)* and *Python* Code *(movieGenre.py)* for the following questions. We recommend that you use another file (*testMovieGenre*.js and *testMovieGenre*.py) to test the functionality of the code written. However, only the '*movieGenre'* files will be evaluated using our testing suite.

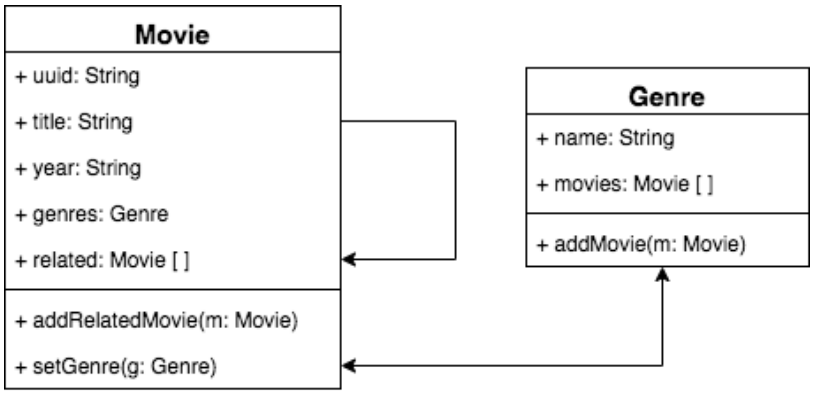

Figure 2 - Movie and Genre Class Diagram

Create the Class for the Movie and the Genre class identified in figure 1 using the following specifications:

1. Create a no parameter **constructor** for the Movie and Genre class. The attributes are configured with reasonable defaults (i.e. empty strings, empty arrays and nulls) in the constructor. [2]
2. The "**addRelatedMovie**" method of the Movie class will check to ensure that an instance of the Movie class is passed as a parameter before adding to the array of Movies called "related". The method will return true if operation is completed successfully. [5]
3. The "**setGenre**" method of the Movie class will check to ensure that an instance of the Genre class is passed as a parameter before changing the genres attribute. The method will also add the current movie object to the genre instance using the "**addMovie**" method of the Genre class. The method will return true if operation is completed successfully. [5]
4. The "**addMovie**" method of Genre class will check to ensure that an instance of the Mobile class is passed as a parameter before addition to the array of Movies called "movies". The movies array is sorted by movie's title. The method will return true if operation is completed successfully and false otherwise. [5]

Section 3 Application

Build a web-based application using the code written in sections 3. In the file app.js and app.html use JavaScript and HTML to complete the following questions:

1. Create a form with the id "**createGenre**" to create a new instance of the Genre class. The form contains fields needed for creating a genre. The fields will have an id with the same name of the field that it captures. The form will capture information from the user and stores the record in a global array called "*genres*" using the "*enqueue*" function from section 3. The form will clear the fields after the record is inserted correctly. After inserting the genre use the "sort" function from section 3, to keep genres in ascending order. [2]

2. Create a form with the id "**createMovie**" to create a new instance of the Movie class. The form contains fields needed for creating a genre. The fields will have an id with the same name of the field that it captures. Association of only one genre is required for the form. The genre is specified via a dropdown list (id = 'genres') which is populated using the "*genres*" global array. The form will capture information from the user and stores the record in a global array called "*movies*" using the "*enqueue*" function from section 3. The form will clear the fields after the record is inserted correctly. [3]

3. Create a function "**displayGenres**" that will use the createElement method to display an ordered list of genres in the section with the id "*genreList*". [2]

4. Create a function "**displayMovies**" that will use strings to display a table of movies in the section with the id "*movieList*". The columns of the tables are related to the fields of the movie class. [2]

5. Write a function called "**loadGenres**" that will make an AJAX GET request to retrieve the file "genreDB.txt" that has a list of genres separated by commas. The function will load the data and populate the global genres array. The *loadGenres* function will also use the *displayGenres* function created in section 4 to display results. [4]

6. Write a function called "**loadMovies**" that will make a request to a json file called "moviesDB.json". The function will load the data and populate the global genres array. The *loadMovies* function will also use the *displayMovies* function created in section 4 to display results. [5]

Section 4 – Data and Declarative Programming

Write **both** *JavaScript (merger.js)* and *Python* Code *(merger.py)* with as many declarative structures (map, filter and reduce) for the following questions. For each question include the unit test *(testMerger.js* and *testMerger.py)* code required to demonstrate the question answered works as expected. However, only the '*merger'* files will be evaluated using our testing suite.

```
['johnny', 'peter' ….]
['bravo', 'griffin', ….]
```

1. Create two arrays "first_names" and "last_names" populated with at least 5 reasonable values and complete the following:
    a. Write a function called "**mergeHandler**" that will accept three parameters; arr1, arr2 and func. [10]
        i. "*func***"** is the function that will be applied on each element in arr1 and arr2. (i.e. arr1[i] and arr2[i] will be supplied to "*func*" for i 0 to n-1)
        ii. "*func*" will return the merged result of two elements
        iii. All the merged results will be stored in an array.
        iv. The **mergeHandler** will return an array with all the elements from *arr1* and *arr2* merged based on the strategy defined in *func*.
        v. Assume that arr1 and arr2 have the same number of elements. (however, you should appreciate how to write the solution to accommodate for multiple sizes of the input)

    b. Write a function called "**merge2Single**" that will use the "*mergeHandler*" to create an array of strings with the first name and last name concatenated into a single string. (produces ['johnny bravo', 'peter griffin']) [5]

```
{
   firstname: "johnny",
   lastname: "bravo"
}
```

    c. Write a function called "**merge2Object**" that will use the "*mergeHandler*" to create an array of objects with the keys '*firstname'* and '*lastname'* populated with the values from the arrays "*first_names*" and "*last_names*". [5]

2. Optional - Not required but useful practise:
3. Optional - In a file called merge.html and useMerge.js, create a UI interface to demonstrate the result of using the three functions created in section 5. [Bonus 12]

Submission

1. Ensure all solution JS files are in the **js** folder, and python in the **py** folder and all test files are in the **test** folder.
2. HTML files are defined in the root of the project
3. All data files “moviesDB.json” and “genreDB.txt” are located in the data folder
4. Provide a file called “members.txt” that will have a list of the id number of group members.

Please note that every addition to the structure of the code to be evaluated will result in a deduction of 5 marks.