**Assignment 2**

**Due Date Thursday, 25 , 2021**

**11: 59 PM**

**Q.1 Basic JS**

1. Consider the following array:

**let *colors*** = [**"white"**, **"blue"**, **"yellow"**, **"black", "red"**, **"green"**]

Using [array destructuring](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) assign the first 2 elements to **firstColor** and **secondColor** variables and assign the remaining elements to **otherColors** variable. Display the values of these 3 variables.

1. Use the [**spread**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax) operator to produce an array named **seasons** by concatenating the following 2 arrays and adding extra elements "Cool!" and "Super Hot Summer!" as seasons.

Input arrays:

**let *cold*** = [**'autumn'**, **'winter'**];  
**let *warm*** = [**'spring'**, **'summer'**];

Output seasons array:  
[**"Cool!"**, **'autumn'**, **'winter'**, **'spring'**, **'summer'**, **"Super Hot Summer!"**]

1. Call the ***Math***.max() function and pass the **nums** array to it. Do not forget to use [**spread**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax) operator to convert the array to multiple arguments expected by the max function.

**let *nums*** = [1, 2, 3, 4, 45, 5, 6]

1. Consider the following the ***square*** function and **nums** array.

**function** *square*(x) {  
 **return** x \*\*2;  
}  
**let *numbers*** = [1, 2, 3, 4, 4, 5, 6, 7, 8, 9, 10];

a. Use the square function along with the array [**map** function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map) to square the elements of **nums** array and assigned the results to **squaredNums**. Then Display **squaredNums**.

b. Enhance the implementation done in previous question by removing the square function and using anonymous [**arrow** function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions).

1. Consider the following array:

**let *nums*** = [11, 22, 33, 46, 75, 86, 97, 98];

* Use [**filter**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter) then [**map**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map) array functions to filter even numbers then square them. Assign the results to a variable named **squaredEvenNums** then display it. The output should be:

squaredEvenNums: **[484, 2116, 7396, 9604**]

* Use the [**reduce**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce) array function to compute the of **sums** array. The output should be:

Sum of array elements: **468**

You must use arrow functions in this exercise.

1. Square and sum the elements of this array using arrow functions and in 1 line of code. Then find the average of the array.

**let *nums*** = [25, 45, 55, 77, 88, 99];

1. Sort the in ascending, descending order [18,47,56,65,47,73,28,18]

**Note** that you must use **JavaScript features and capabilities** such as **arrow functions**, array functions **(.map, .reduce, .filter, .splice, .sort…) and** **spread operator**.

**Q2. JS OOP [ONLY SOLVE ONE OF THE QUESTION]**

**OPTION 1.****– Banking App**

In this exercise you will build a simple banking system according the design shown in Figure 1.

Diagram

Description automatically generated

Figure 1. Banking System Class Diagram

1. Create **BankAccount** class with the following properties: accountNo, balance. This class should have a constructor to initialize these 2 properties. This class should have these methods:
   * deposit(amount): this method adds the amount to the balance
   * withdraw(amount): this method subtracts the amount from the balance
   * toString(): this method return Account # **accountNo** has QR **balance**. e.g., Account #123 has QR1000.

Export the **BankAccount** class module as an object.

1. Create app.js program. Declare **accounts** variable array and initialize it with the following accounts:

|  |  |
| --- | --- |
| **accountNo** | **balance** |
| 123 | 1000 |
| 234 | 4000 |
| 345 | 3500 |

Display the content of the **accounts** array.

1. Create **SavingAccount** class that extends BankAccount with an extra property: minBalance and an extra method distributeBenefit(benefitPercentage). This method computes the monthly benefit using the balance += (balance \* benefitPercentage). The constructor should extend BankAccount to initialize the minBalance. Also, extend the toString() to indicate that this is a Saving Account. e.g., e.g., **Saving** Account #123 has QR1000.

Test savingAccount in app.js using the same table above and use a minimum balance of 500 for all accounts.

1. Create **CurrentAccount** class that extends BankAccount with an extra property: monthlyFee and an extra method deductFee(). This method subtracts the monthlyFee from the account balance only if the deducted fee is less than the balance. The constructor should extend BankAccount to initialize the monthlyFee. Also, extend the toString() to indicate that this is a Current Account. e.g., e.g., **Current** Account #123 has QR1000.

Test currentAccount in app.js using the same table above and use a monthly fee of 15 for all accounts.

1. Create **Bank** class to manage accounts. It should have a property **accounts** to store the accounts. Also, it should have the following methods:

|  |  |
| --- | --- |
| **Method** | **Functionality** |
| add(account) | Add account (either Saving or Current) to accounts array. |
| getAccount(accountNo) | Return an account by Id |
| deleteAccount(accountNo) | Delete an account by Id |
| avgBalance() | Get the average balance for all accounts |
| sumBalance() | Get the sum balance for all accounts |
| serialize() | Return accounts as a JSON string |
| deserialize(accountsJson) | Takes JSON string representing accounts and returns an array of accounts. |

1. Create app.js program. Declare an instance of Bank class then add the following accounts:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **accountNo** | **balance** | **type** | **minimumBalance** | **monthlyFee** |
| 123 | 500 | Saving | 1000 |  |
| 234 | 4000 | Current |  | 15 |
| 345 | 35000 | Current |  | 25 |
| 456 | 60000 | Saving | 1000 |  |

1. Test all the bank methods described above.
2. Display the total balance of all accounts.
3. Go through all the **Current** accounts and charge the monthly fee
4. Display the total balance of all accounts after charging the monthly fee.
5. Go through all the **Saving** accounts and distribute the benefit using a 5% benefit.
6. Display the total balance of all accounts after distributing the benefits.



**OPTION 2– Book Donation App**

You are required to develop a book donation App that allows people to donate their unused books to people who needs it.

The book donation system should have the following classes

1. **Book** – bookId, title, author, imageUrl, donor, status

Donor property is an object having of the details of the person who donated the book. Note that the **status** attribute can have one of the following values:

* + **Pending** : As soon as the user adds the book s/he want to donate to the system, the status is set to pending
  + **Available** : When the book donor delivered the book to the store then status of the book becomes available

1. **Donor** - donorId, firstname , lastname , phoneNumber , street, city, email, password
2. **BookCatalog** – contains list of books and the following methods

|  |  |
| --- | --- |
| **addBook** | Input: book object.  Adds a book to the list of books. All books are pending when they are first added. |
| **updateBook** | Input: book. Updates the book that having matching bookId. |
| **deleteBook** | Input: bookId. Deletes the book from the list of books. |
| **getBooks** | Returns all the books that are not pending. |
| **getDonorBooks** | Input donorId. Returns all the books donated by a particular donor. |
| **getTopXDonors** | Input: donorsCount. Returns the top book donors. Eg. If the user passes 3 as donorsCount parameter then this function returns the top 3 donors and the list of books each one donated. |

 You do not need to create **Donor** and **Book** classes. Just create the objects directly. And test your code by creating **app.js file** to instantiate the **BookCatalog** class and test its methods.

**Note** that you should make use of the **JavaScript features and capabilities** such as **arrow functions**, array functions **(.map, .reduce, .filter, .splice, .sort…),** **spread operator**, **object literals**, **classes**.

**Q.4 Basic JS Unit Testing Using Mocha and Chai**

1. Open ***UnitConverter project*** in Webstorm. You should see a JavaScript file named *UnitConverter.js*. In this exercise, you will create a spec file to unit test the function of the *UnitConverter* class.
2. Write a unit test for each of the methods mention in the below table

You may start with the following inputs and expected results. Then use search for “google unit converter” to compute the expected results for more input values.

|  |  |  |
| --- | --- | --- |
| **Method** | **Input** | **Expected Result** |
| kgToOunce | 1 | 35.274 |
| kgToPound | 2 | 4.4092 |
| meterToInch | 1 | 39.3701 |
| meterToFoot | 2 | 6.5617 |

9. Run the unit tests as you develop them using WebStorm:

Graphical user interface, text, application, website

Description automatically generated

After you complete the assignment, fill in the ***TestingDoc-Grading-Sheet.docx***and save it inside ***Assignment2*** folder. Push your work to Github repository.