Lesson 1 Assignment

In this lesson you will be developing functions to find the intersection, relative compliment, union and symmetric difference between two sets. Let's take a look at the process you will need to use to develop these functions. This will serve as a model for developing the other three set functions.

Today, good developers use a test driven development process to create and test their functions. We will use this process to develop the four set functions for this lesson. The process consist of the following steps:

1. Define the problem (the inputs, outputs, task and validation rules)
2. Define a test matrix (test cases to prove that the function is working correctly)
3. Define an algorithm for the function
4. Test the algorithm against the test matrix
5. Translate the algorithm in to code
6. Run all of your test cases to prove that the function is working correctly
7. Deploy your code to the repository

Let's go through an example of this process to develop the `intersection()` function. Follow the steps below.

Step 1:

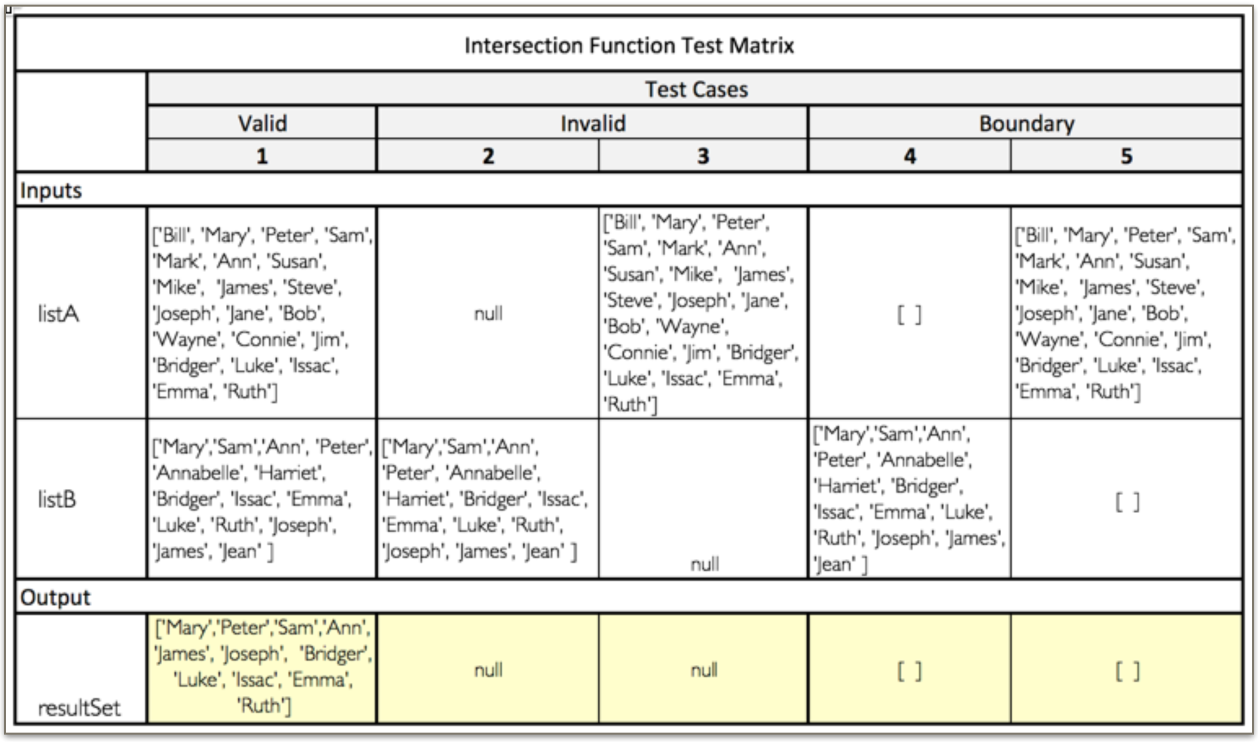
First, we define the problem. When we define the problem we determine four things. What are the inputs? What will be output? What is the task? What validation rules must be enforced.

In the intersection function, the inputs will be the drug A list and the drug B list that we will be finding the intersection of. The output will be a list containing the result. The task is to find the intersection of two sets. The intersection is the list of those items in both list that are in both List A and List B. What validation rules should be enforced? Normally, this means defining the validation rules for valid inputs and any any other special business rules that need to be enforced. The only time that the two inputs, list A and list B, will be invalid is when the person calling the program fails does not pass either list A and/or list B to the function.

Step 2:

In Step 2 of the process, we define test cases that be used to prove that the function is working as specified. We define test cases for when valid inputs are entered, invalid inputs are entered and when valid inputs stress the limits of the function. The test cases are often done in a test matrix. The test matrix defines the inputs and the expected output for each test case to be run. This test matrix in a sense defines the exact behavior of the function for all valid, invalid and boundary conditions. It becomes a contract for what the function will do. Your function is not complete until all of the test cases complete successfully when you call the function.

Here is the test matrix for the intersection function.



Intersection Function Test Matrix

	Test Cases				
	Valid	Invalid		Boundary	
	1	2	3	4	5
Inputs					
listA	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	null	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	[ ]	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']
listB	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	null	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	[ ]
Output					
resultSet	['Mary','Peter','Sam','Ann', 'James', 'Joseph', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	null	null	[ ]	[ ]

We first defined a test case for typical valid inputs. Then we defined two test cases for invalid inputs. Test 2 defines that the output of the function will be `null` when a `listA` is `null`. If the input parameter is null, it means that no list was passed to the function for `listA`. Similarly, Test 3 defines what happens when `ListB` is null. Finally, Test 4 and Test 5 define what happens when `ListA` and/or `ListB` is empty. They passed you a list but the list has no elements in the list. This is analogous to someone giving you a container or box that has nothing in it.

Step 3:

In Step 3 of the functional development process, we define an algorithm for the function. An algorithm is a list of instructions that need to be performed to accomplish the main task of the function. In most cases, functions automate a real task that a person would do in real life; therefore, it makes sense for us to first describe the steps that a person would take to accomplish the same task.

The first line of an algorithm is always the function signature. It defines name of the function, the list of inputs to the function and the output value to be returned. The basic syntax for a function signature is as follows:

```
functionName( input1, input2, …): outputValue
```

The function signature is usually followed by the `BEGIN` and `END` keywords to indicate the beginning and end of statements inside the function. Here is the function signature for the intersection function.

```
intersection(listA, listB): resultSet
BEGIN
  … instructions go here …
END
```

First we stated the name of the function (`intersection`). We then went to the test matrix created in the previous step and listed the two inputs (`listA` and `ListB`) defined in the test matrix within the parenthesis. The output value (`resultSet`) defined in the test matrix is specified after the colon at the end of the function signature.

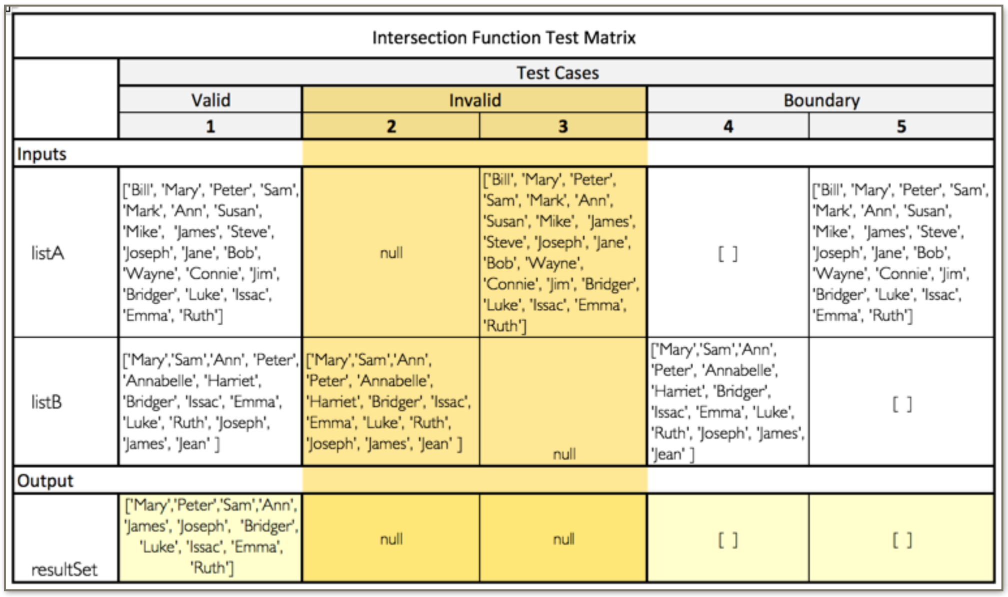
The main body or list of instruction defined in an algorithm generally follow the same outline.

```
intersection(listA, listB): resultSet
BEGIN
     // If the inputs are invalid then
        // return errorCode

     // Instructions to perform the main task
        …
        …

     // return the output value
END
```

1. Check to see if the inputs are invalid. If an input is invalid, return a value to indicate that an error occurred and exit the function,
2. Perform the steps to accomplish the main task, and finally
3. Return the output value and exit the function.

Checking for invalid inputs

The first part of the algorithm should always check for invalid inputs. The test matrix list all of the different cases where invalid inputs can be entered.



Intersection Function Test Matrix					
	Test Cases				
	Valid	Invalid		Boundary	
	1	2	3	4	5
Inputs					
listA	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	null	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	[ ]	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']
listB	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	null	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	[ ]
Output					
resultSet	['Mary','Peter','Sam','Ann', 'James', 'Joseph', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	null	null	[ ]	[ ]

Test cases 2 and 3 in the intersection test matrix define that `ListA` and `ListB` are invalid when their value is `null`. We need to write a statement to test to see if either List A is `null` or List B is `null`. If either condition evaluates to true then we should exit the function and return a null to indicate that an invalid input was entered.

```
intersection(listA, listB): resultSet
BEGIN
     // if ListA is null or ListB is null then
        // return - 1

     // Instructions to perform the main task …

     // return the result …
END
```

Defining main task

Next, we need to define the instructions to accomplish the main task of the function (finding the intersection of `ListA` and `ListB`).

A function normally simulates what a person would do to perform the specified task. Let's take a look the actual steps taken by an person to find the intersection or list of those items that are in common between two list. Select the link below to watch the video illustrating how this can be done.

Steps taken by a person to find the intersection of two list

Here are the list of instructions that we documented to find the intersection of two list.

```
     create a new resultSet list

     for every item in listA
        currentItemA = get next name from listA

        for every item in listB
           currentItemB = get next name from ListB
           if (currentName == current item in listB) then
              add currentName to end of resultSet list
              break out of (or end) the listB loop
        end listB loop

     end listA loop
```

The algorithm for the intersection function now looks like this after adding these instructions and the return statement to the function.

```
intersection(listA, listB): resultSet
BEGIN
    if ListA is null or ListB is null then
      return NULL

    // Instructions to perform the main task …
    create a new resultSet list

    for every item in listA
      currentItemA = get next name from listA

      for every item in listB
        currentItemB = get next name from ListB
        if (currentName == current item in listB) then
          add currentName to end of resultSet list
          break out of (or end) the listB loop
      end listB loop

    end listA loop

    // return the result
    return resultSet
END
```

Let's take a closer look at each of the instructions in the algorithm. The first statements checks to see if the two input lists are invalid. If either of the two lists are equivalent to `null` (meaning that they reference nothing and are invalid), a `null` output value is returned to signal that an error occurred.

If the inputs are valid, control then moves to the next statement and a new empty resultSet. Next, we loop through every item in `listA.`We get the next item in `ListA` and assign it to the `currentName` variable. Then we loop through every item in the `listB` list. We assign the next item in `listB` to the `currentItemB` variable. If the next item in `listB` is equal to the `currentName`, we add it to the end of the `resultSet` list and break out of (or exit) the `listB` inner loop. If `currentItemA` and `currentItemB` are not equivalent, we continue repeat the `listB` inner loop and get the next item in `listB`. This loop continues until either a match is found or we get to the end of the list. In either case control returns back to the outer `listA` outer loop and we repeat the process for the next item in `listA`.. This process continues until we have exhausted all of the elements in `listA`.

Finally, we return the `resultSet` list.

Step 4:

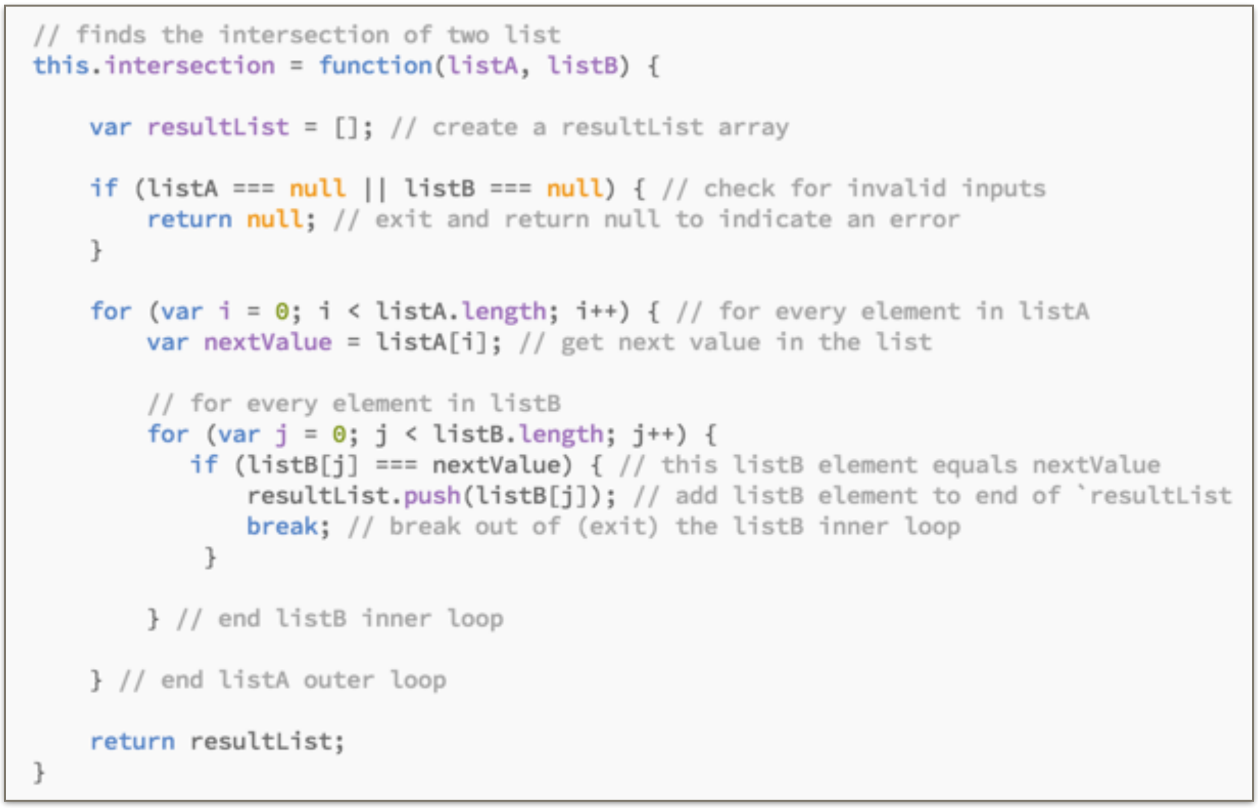
Step 4 of the development process is to test the function before we start writing any code. We run a test for each test case in the test matrix to verify that the logic of the function is returning the expected output values. Testing your algorithm saves time and money. It is much cheaper to fix an error in our logic at this point than when we have written a bunch of code.

You test an algorithm by playing computer for each test case defined in the test matrix. For each test case, you define memory locations for each of the variables defined in your algorithm. Then you assign the input values defined in the test case to the memory locations corresponding to the input parameter variables. Then you go through each instruction line-by-line in the algorithm from top to bottom pretending that you are the computer executing each instruction. After executing all of the instructions in your algorithm, check to see if the result returned matches the output defined in the test matrix for that test case. If the result matches that of the output defined in the test matrix then the test passes. If the results do not match then your logic is not correct and you will have to fix your algorithm and rerun all of the test cases in the test matrix. Your function is working as expected when all of the test cases defined in the test matrix pass successfully.Select the link below to watch a demonstration of how to test an algorithm.

Testing an algorithm

Step 5:

The fifth step in the development process is to translate each instruction of the algorithm into code. The implementation of the intersection function in JavaScript is generally straight forward. The intersection function can be used as a template to implement of the three other functions so be sure that you understand each statement in the function. If you do not understand any of the instructions in the code below, go to the W3Schools JavaScript tutorial or one of the other suggested resources to learn more about JavaScript and the specific commands you do not understand.



```
// finds the intersection of two list
this.intersection = function(listA, listB) {

    var resultList = []; // create a resultList array

    if (listA === null || listB === null) { // check for invalid inputs
        return null; // exit and return null to indicate an error
    }

    for (var i = 0; i < listA.length; i++) { // for every element in listA
        var nextValue = listA[i]; // get next value in the list

        // for every element in listB
        for (var j = 0; j < listB.length; j++) {
           if (listB[j] === nextValue) { // this listB element equals nextValue
               resultList.push(listB[j]); // add listB element to end of `resultList
               break; // break out of (exit) the listB inner loop
            }

        } // end listB inner loop

    } // end listA outer loop

    return resultList;
}
```

Set up the development environment

You will need an IDE (In to create all of the files for your class. A good editor that supports HTML, CSS, Java Script and Angular 2 is called WebStorm. You also need to have a code repository. You will be using Git to submit your assignments and to share your code.

Select the link below to set up the development environment for your project.

Set up the development environment

Create your functions

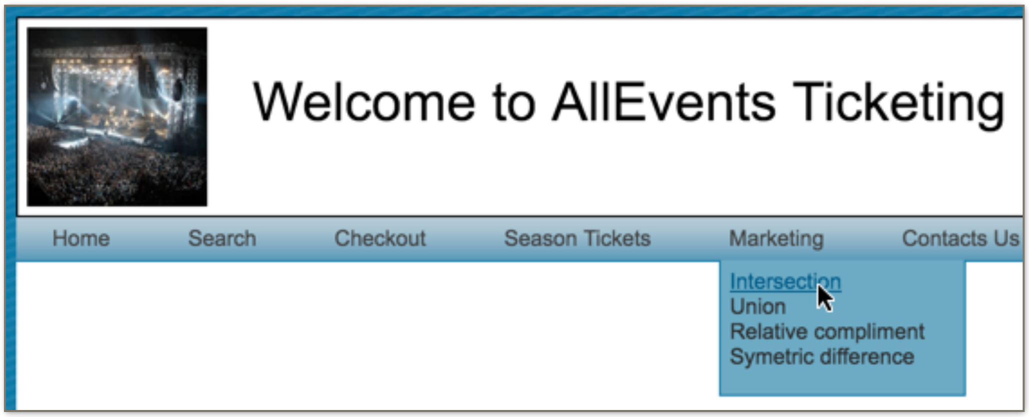
1. Download and unzip the `AllEventsTicketing.zip` compressed file. Copy the uncompressed `AllEventsTicketing` folder to your `WebStorm/CIT366Projects` folder.

2. Open the uncompressed `AllEventsTicketing` folder in the WebStorm IDE.

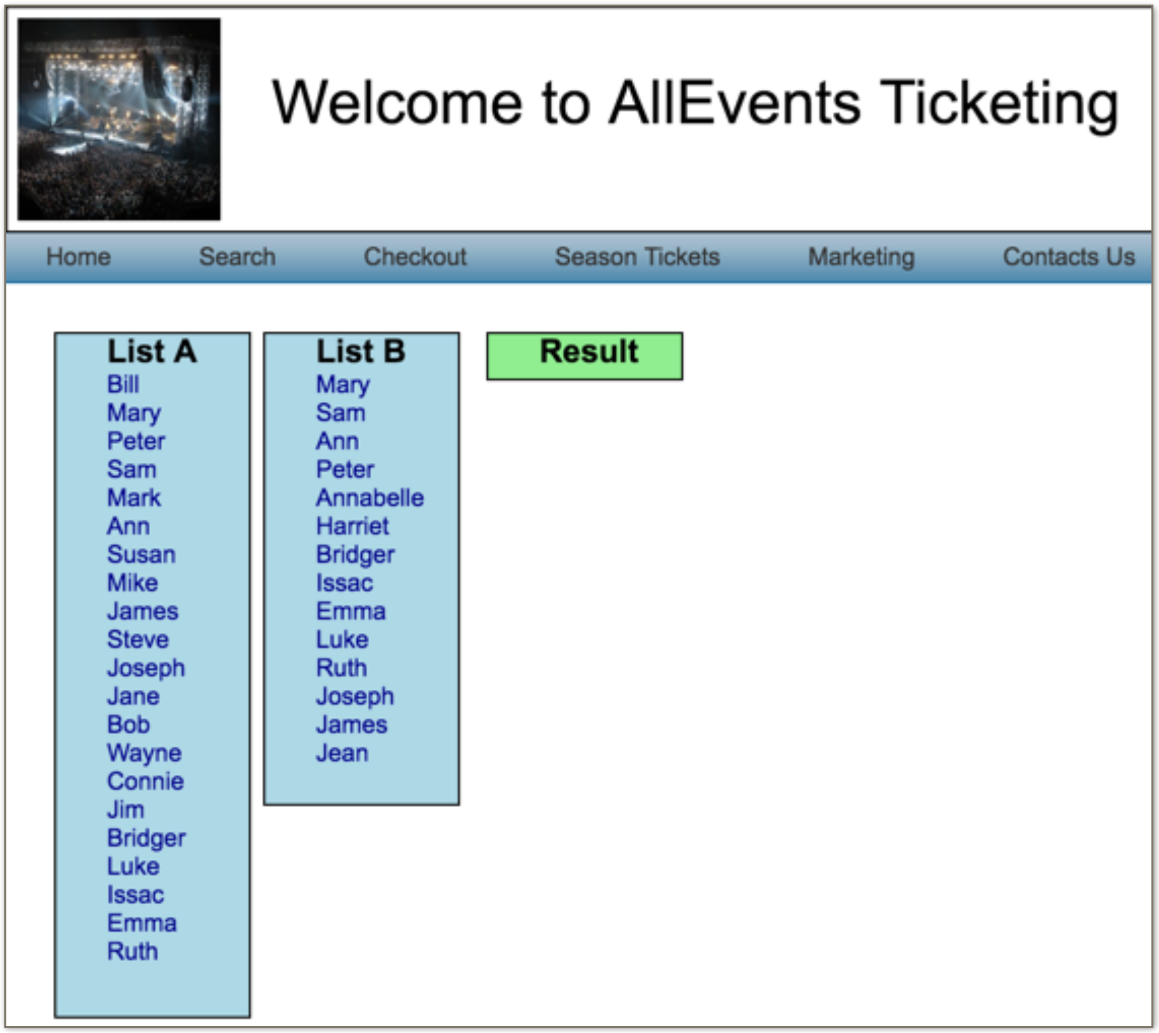
   This folder contains all of the files for a small website for a business called **Home Town Bank**.

3. Display the `index.html` file in a Chrome browser window. This is done by starting the Chrome browser and then opening the file the `index.html` file in the `WebStorm/CIT366Projects` folder.

4. Select the Marketing Analysis menu item. The four commands allow the analyst to do marketing research on what types of tickets their customers are buying. Select the **Intersection**, command.

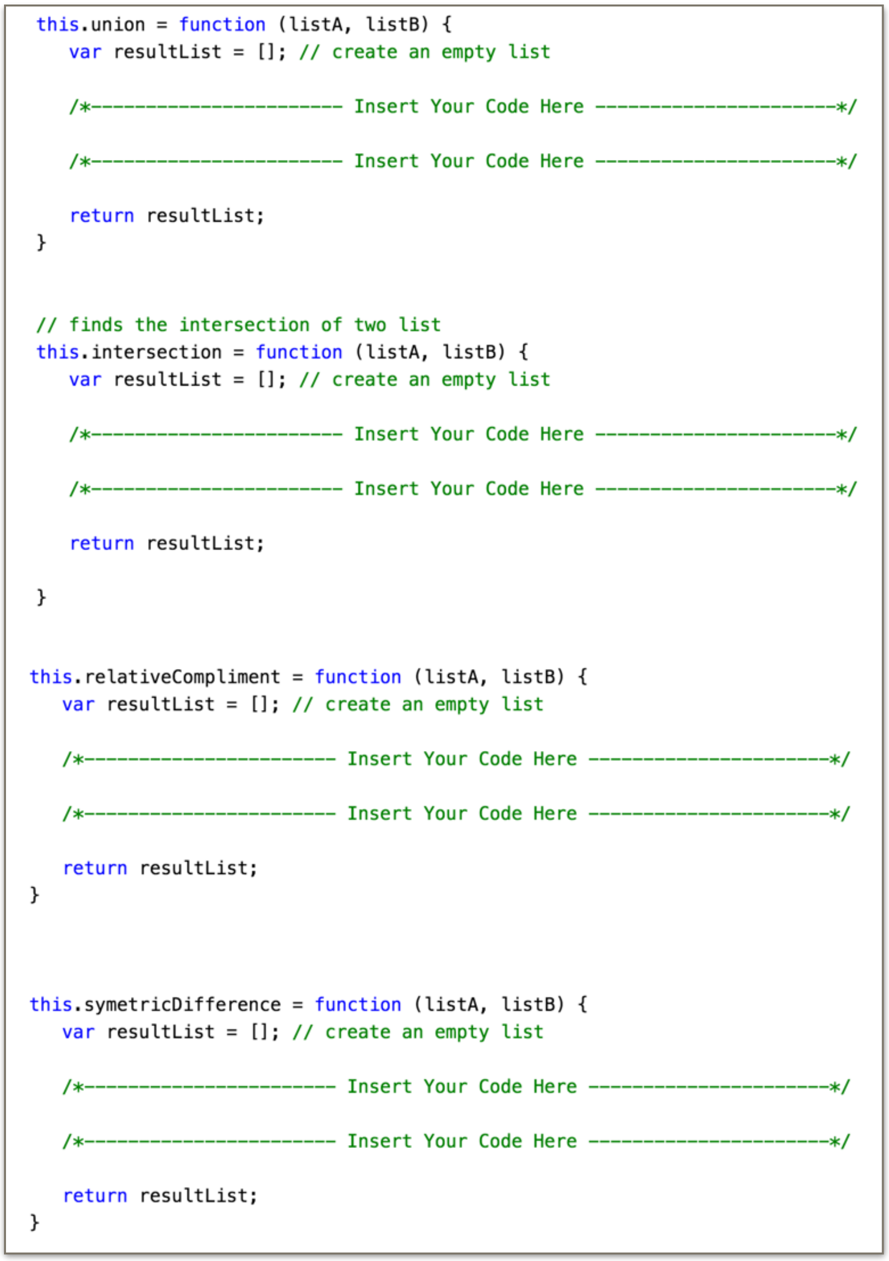


Two list of names will display (`ListA` and `ListB`). Each these list represents a different set of customers. A third empty `Result` list is also displayed. This list is empty because the `intersection()` set function has not yet been created. The results of calling the function will be displayed here once you successfully implement the function.



List A	List B	Result
Bill	Mary	
Mary	Sam	
Peter	Ann	
Sam	Peter	
Mark	Annabelle	
Ann	Harriet	
Susan	Bridger	
Mike	Issac	
James	Emma	
Steve	Luke	
Joseph	Ruth	
Jane	Joseph	
Bob	James	
Wayne	Jean	
Connie		
Jim		
Bridger		
Luke		
Issac		
Emma		
Ruth		

5. The four intersection functions are to all be implemented in the `Set.js` file. Switch back to the WebStorm IDE and open the `Set.js` file. Scroll down to the bottom of the file and you will see the four functions that you need to implement for this lesson.



```
 this.union = function (listA, listB) {
    var resultList = []; // create an empty list

    /*---------------------- Insert Your Code Here ----------------------*/

    /*---------------------- Insert Your Code Here ----------------------*/

    return resultList;
 }


 // finds the intersection of two list
 this.intersection = function (listA, listB) {
    var resultList = []; // create an empty list

    /*---------------------- Insert Your Code Here ----------------------*/

    /*---------------------- Insert Your Code Here ----------------------*/

    return resultList;

 }


this.relativeCompliment = function (listA, listB) {
   var resultList = []; // create an empty list

   /*---------------------- Insert Your Code Here ----------------------*/

   /*---------------------- Insert Your Code Here ----------------------*/

   return resultList;
}



this.symetricDifference = function (listA, listB) {
   var resultList = []; // create an empty list

   /*---------------------- Insert Your Code Here ----------------------*/

   /*---------------------- Insert Your Code Here ----------------------*/

   return resultList;
}
```

Each of these functions have two input parameter variables, `listA` and `listB`. These variables will respectively contains the two arrays (list) that you will need to perform the set operation on. The first statement in the function block defines a variable called `resultList` and assigns an empty array (list) to it. The last statement of each function returns the `resultList` array. The two block comments indicate where you will need to insert the code to implement each of these functions.

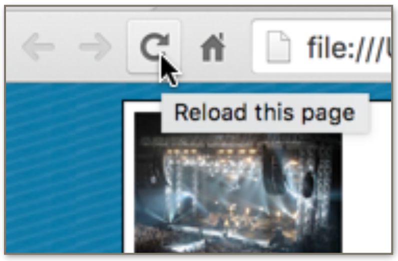
Start by implementing the `intersection()` function. The code that you need to enter is shown below. Make sure you understand every line of code and what it does, or you will not be able to implement the other functions. If you do not what an line is code is doing you will need to study and/or ask to found out.

```
// finds the intersection of two list
this.intersection = function(listA, listB) {

    var resultList = []; // create a resultList array

    if (listA === null || listB === null) { // check for invalid inputs
        return null; // exit and return null to indicate an error
    }

    for (var i = 0; i < listA.length; i++) { // for every element in listA
        var nextValue = listA[i]; // get next value in the list

        // for every element in listB
        for (var j = 0; j < listB.length; j++) {
            if (listB[j] === nextValue) { // this listB element equals nextValue
                resultList.push(listB[j]); // add listB element to end of resultList
                break; // break out of (exit) the listB inner loop
            }
        } // end listB inner loop
    } // end listA outer loop

    return resultList;
}
```

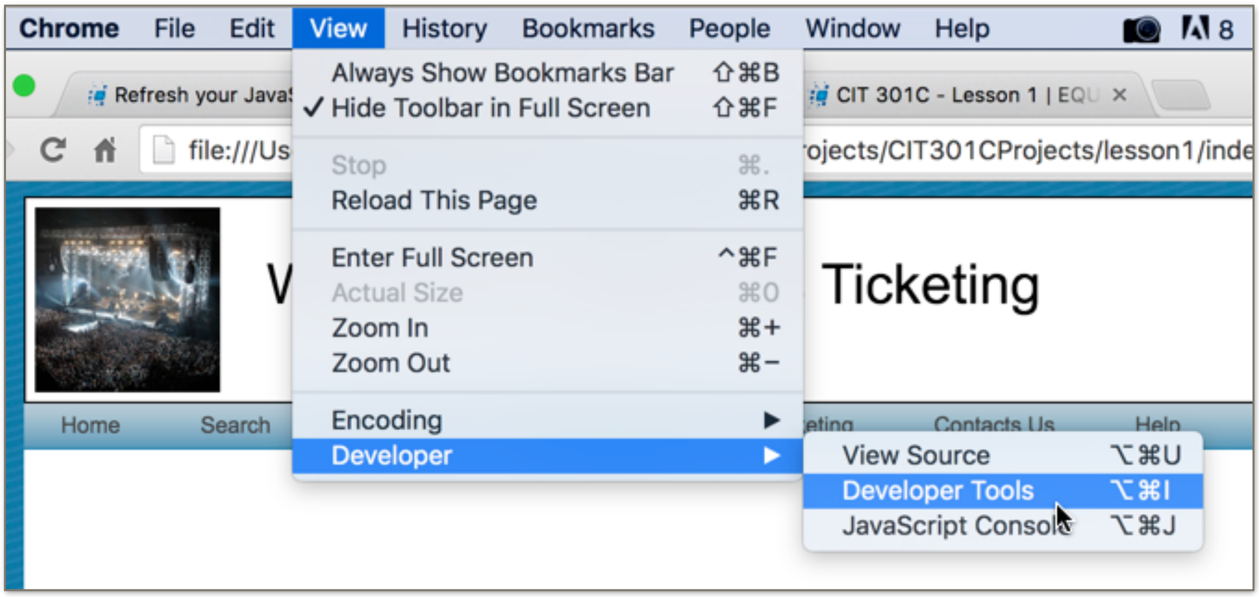
6. Save the file when you are done.

Checking and fixing syntax errors

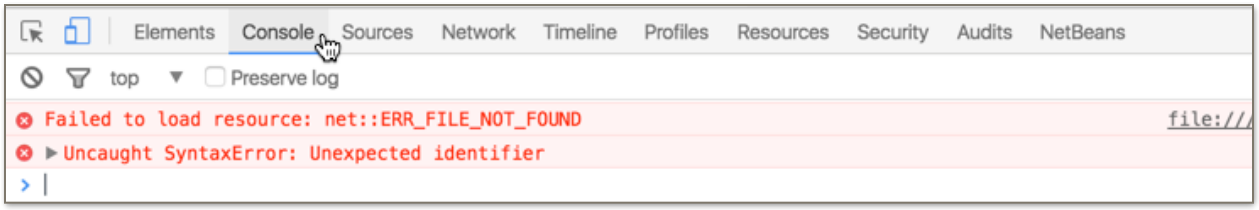
Switch back to the browser to view the AllEventsTicketing home page. Reload the webpage and all of the latest changes in your code to the browser.



Open the Developer Tools window by selecting the **View**, **Developer**, **DeveloperTools** command.

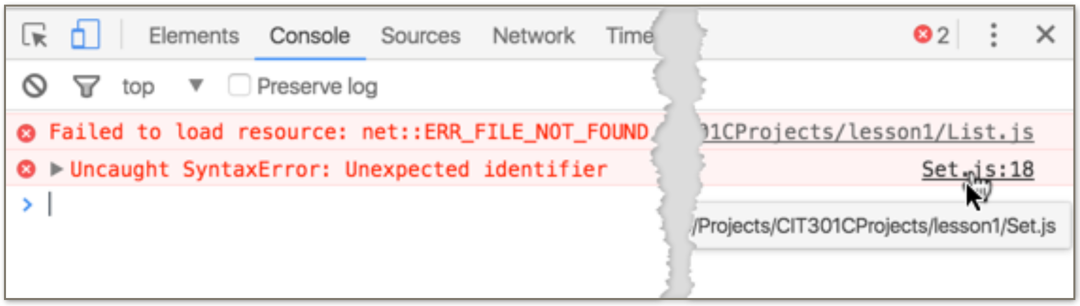

Chrome File Edit View History Bookmarks People Window Help
Always Show Bookmarks Bar ⇧⌘B
✓ Hide Toolbar in Full Screen ⇧⌘F
Stop ⌘.
Reload This Page ⌘R
Enter Full Screen ^⌘F
Actual Size ⌘0
Zoom In ⌘+
Zoom Out ⌘−
Encoding
Developer
View Source ⌥⌘U
Developer Tools ⌥⌘I
JavaScript Console ⌥⌘J
Refresh your Java
CIT 301C - Lesson 1 | EQU
file:///Us
ojects/CIT301CProjects/lesson1/inde
Ticketing
Home Search Contacts Us Help

Select the **Console** tab and check to see if there are any syntax errors that need to be fixed.


Elements Console Sources Network Timeline Profiles Resources Security Audits NetBeans
top
Preserve log

```
Failed to load resource: net::ERR_FILE_NOT_FOUND                    file:///
▶ Uncaught SyntaxError: Unexpected identifier
```

If there is an error message, then select the hyperlink to the file listed to the right of the error message. This will take you to the line in the file where the error was detected.



```
Failed to load resource: net::ERR_FILE_NOT_FOUND   01CProjects/lesson1/List.js
▶ Uncaught SyntaxError: Unexpected identifier                   Set.js:18
                                   /Projects/CIT301CProjects/lesson1/Set.js
```

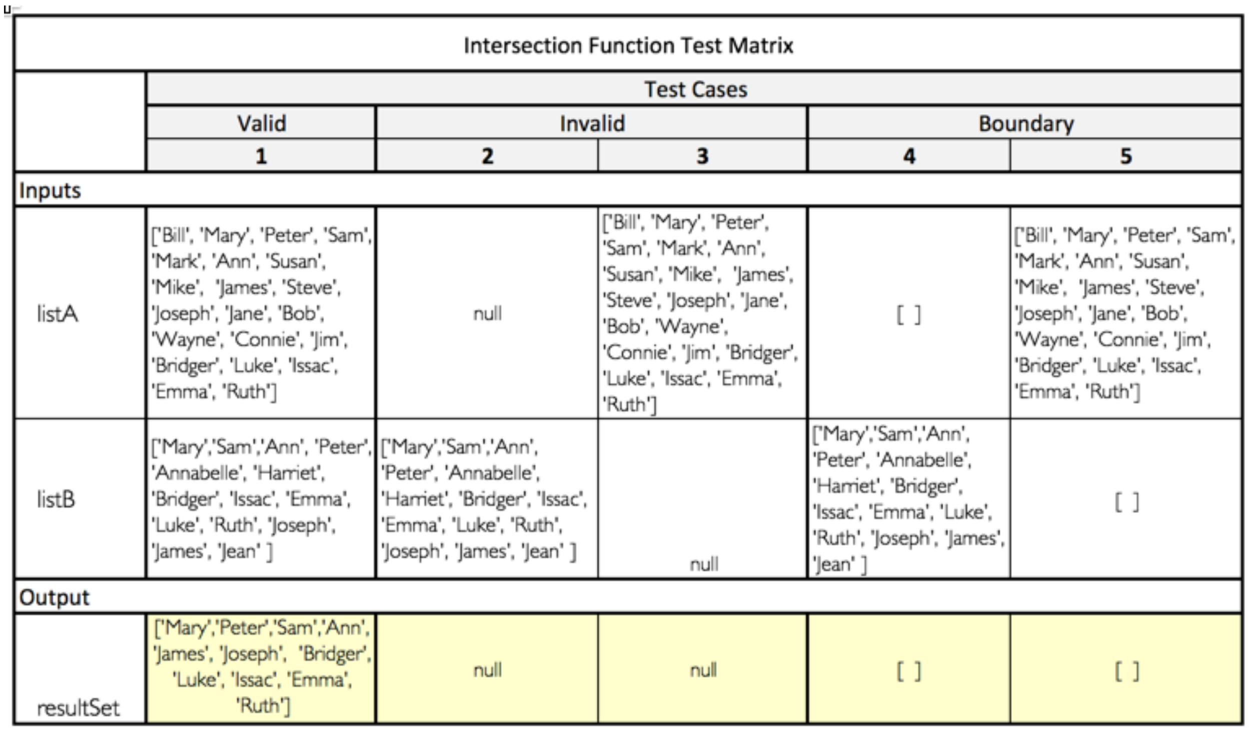
Open up the `Set.js` file in WebStorm IDE, fix the error, save the file, reload the browser and check the console for any additional errors. Repeat this process until there are no more errors.

Step 6:

The sixth step in the function development process is to actually call and run the intersection function developed for each of the test cases defined in the test matrix.

Unit test your functions using QUnit.

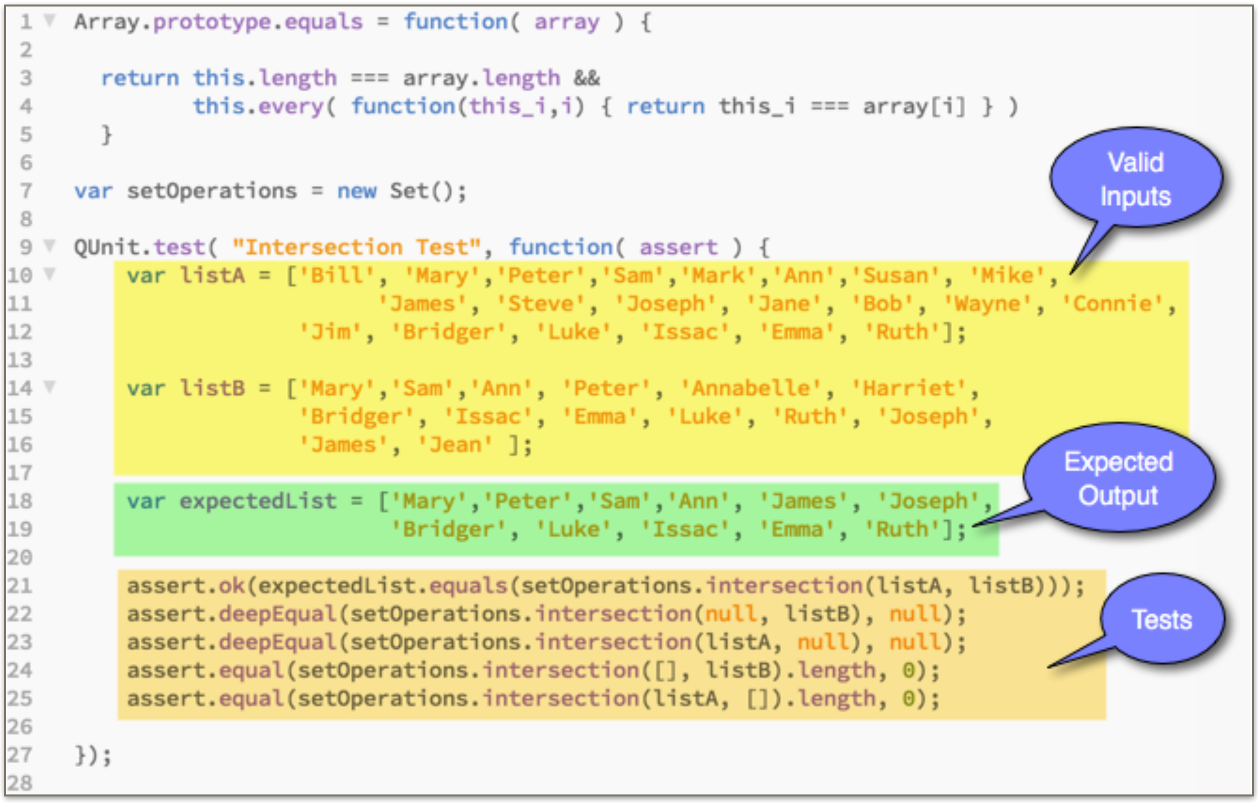
QUnit is a special JavaScript framework that you can download from the Internet. With it we can create a unit test program that will run and test each of the test cases defined in the test matrix.



Intersection Function Test Matrix					
	Test Cases				
	Valid	**Invalid**		**Boundary**	
	1	**2**	**3**	**4**	**5**
Inputs					
listA	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	null	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	[ ]	['Bill', 'Mary', 'Peter', 'Sam', 'Mark', 'Ann', 'Susan', 'Mike', 'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie', 'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']
listB	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	null	['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet', 'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph', 'James', 'Jean']	[ ]
Output					
resultSet	['Mary','Peter','Sam','Ann', 'James', 'Joseph', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth']	null	null	[ ]	[ ]

To use QUnit, you must first create a program that defines each of the test cases to be run, and then create a special html file to launch and run the test program.

The test program for the `intersection()` function and the three other set operation functions was created for you in the `unitTest.js` file in your project. Open the `unitTest.js` file in the WebStorm IDE and examine it. The unit test for the `intersection()` function are shown below.



```
Array.prototype.equals = function( array ) {

  return this.length === array.length &&
         this.every( function(this_i,i) { return this_i === array[i] } )
  }

var setOperations = new Set();

QUnit.test( "Intersection Test", function( assert ) {
    var listA = ['Bill', 'Mary','Peter','Sam','Mark','Ann','Susan', 'Mike',
                      'James', 'Steve', 'Joseph', 'Jane', 'Bob', 'Wayne', 'Connie',
                'Jim', 'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth'];

    var listB = ['Mary','Sam','Ann', 'Peter', 'Annabelle', 'Harriet',
                'Bridger', 'Issac', 'Emma', 'Luke', 'Ruth', 'Joseph',
                'James', 'Jean' ];

    var expectedList = ['Mary','Peter','Sam','Ann', 'James', 'Joseph',
                       'Bridger', 'Luke', 'Issac', 'Emma', 'Ruth'];

    assert.ok(expectedList.equals(setOperations.intersection(listA, listB)));
    assert.deepEqual(setOperations.intersection(null, listB), null);
    assert.deepEqual(setOperations.intersection(listA, null), null);
    assert.equal(setOperations.intersection([], listB).length, 0);
    assert.equal(setOperations.intersection(listA, []).length, 0);

});
```

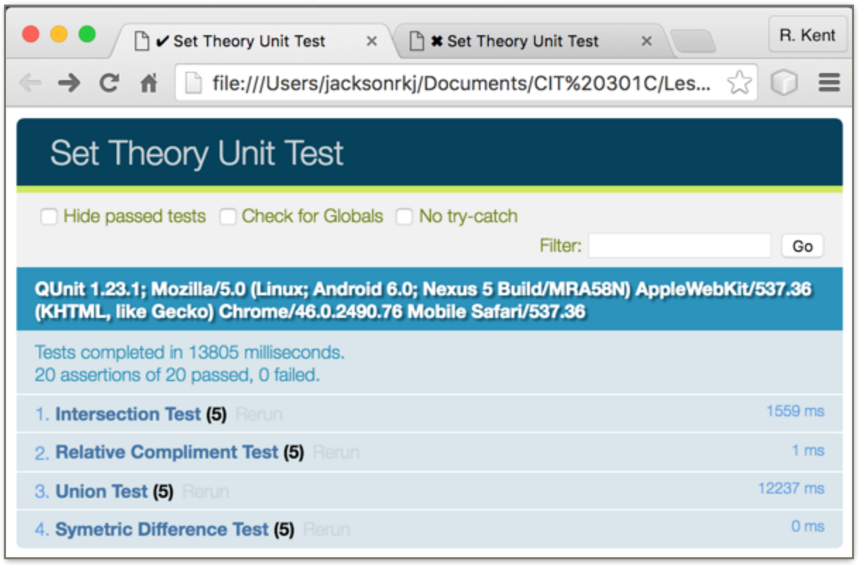
Each call to the `QUnit.test()` function in the file test one of the set operation functions. The first argument passed to the second argument passed to the `QUnit.test()` function contains a short textual description of the test. The second argument is a function that will be automatically called to run each of the unit test.

The function passed as an argument to the `QUnit.test()` function in the code above first defines two variables, `listA` and `listB`, that will be passed as inputs the `intersection()` function. The third variable defines the expected list of values to be returned when the `intersection()` function is called. After that are a series assert statements. Each assert statement runs one of the unit test defined in the test matrix by first calling the `intersection()` function with the inputs defined in the test matrix for that test case and then comparing the returned results with the expected results to see if they are equal. If the values are equal then the test case passed; otherwise, the test case will be marked as failed.

To run the test cases, you must also create a special html web page. Open the `unitTest.html` file in WebStorm. This web page must includes `<script>` tags to reference the `qunit-1.23.1.js` test library, the JavaScript file (`set.js`) that contains the functions to be tested and the JavaScript file (`unitTest.js`) that contains the unit test program.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Set Theory Unit Test</title>
    <link rel="stylesheet" href="https://code.jquery.com/qunit/qunit-1.23.1.css">
</head>

<body>
    <div id="qunit"></div>
    <div id="qunit-fixture"></div>
    <script src="https://code.jquery.com/qunit/qunit-1.23.1.js"></script>
    <script src="Set.js"></script>
    <script src="unitTest.js"></script>
</body>

</html>
```

To run the test program, you simply open this web page (`unitTest.html`) in the Chrome browser. The unit test program will automatically be run and a report will then be displayed indicating which test cases passed and failed.



Set Theory Unit Test

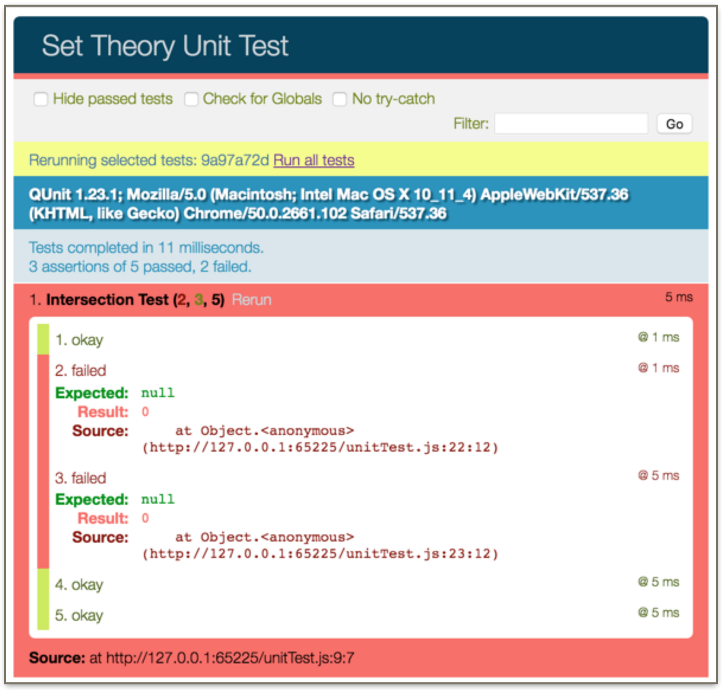
QUnit 1.23.1; Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.76 Mobile Safari/537.36

Tests completed in 13805 milliseconds.
20 assertions of 20 passed, 0 failed.

1. Intersection Test (5) 1559 ms
2. Relative Compliment Test (5) 1 ms
3. Union Test (5) 12237 ms
4. Symetric Difference Test (5) 0 ms

Here the report indicates that all of the test cases passed successfully.

Here is the report when one or of the test cases fails.



Set Theory Unit Test

Hide passed tests Check for Globals No try-catch

Filter: Go

Rerunning selected tests: 9a97a72d Run all tests

QUnit 1.23.1; Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36

Tests completed in 11 milliseconds.
3 assertions of 5 passed, 2 failed.

```
1. Intersection Test (2, 3, 5) Rerun                                5 ms
  1. okay                                                        @ 1 ms
  2. failed                                                      @ 1 ms
  Expected: null
    Result: 0
    Source:     at Object.<anonymous>
            (http://127.0.0.1:65225/unitTest.js:22:12)
  3. failed                                                      @ 5 ms
  Expected: null
    Result: 0
    Source:     at Object.<anonymous>
            (http://127.0.0.1:65225/unitTest.js:23:12)
  4. okay                                                        @ 5 ms
  5. okay                                                        @ 5 ms
Source: at http://127.0.0.1:65225/unitTest.js:9:7
```

Step 7:

Step 7 is the last step of the function development process. It involves deploying your code in your local Git repository and then pushing it up to your remote GitHub repository.

1. You next need to commit all of the files of your project to your local Git repository. This backs up your files and puts them under version control. You do this by first typing in a date and a descriptions of the changes that you have made in the comment text field. Then elect the check mark at the top of the window. This will commit all of the files listed below to your local repository.

2. Push your committed changes to you're remote GitHub repository. Select the **...** menu again and select the **Push** command.

   You have successfully committed your changes to your local Git repository and then pushed the changes up to your remote GitHub repository if not errors occurred.

3. Select the Explore button on the side bar to view all of the files in the folder.

Homework Assignment

Use the development process above to develop the test matrix, algorithm and code for the `relativeCompliment()`, `union()` and `sysmetricDifference()` functions.

1. Make sure you understand what each of the three set operations functionally does. Review the videos and reading assignments if needed.

2. For each of the three functions:

   a. Develop a test matrix similar to the one created for the intersection() function above. The input values may be the same but you will need to define the output list for each of the valid, invalid and boundary test cases.

   b. Develop the algorithm for the function and test it by playing computer for each of the unit test cases.

   c. Open the `Set.js` function in WebStorm and translate the algorithm into JavaScript code.

   d. Select the `unitTest.html` file in WebStorm and select the Live Preview button to run the test cases for the function. If one of the test cases fails, you will need to debug to find the problem, fix the code and then rerun your test cases.

> **Hint**: First implement and test the `intersection()` and `relativeCompliment()` functions. These functions can then be called and used to find the union and symmetric difference of two sets. Review the <u>Set Theory</u> video on how this can be done by combining set operations.

3. **Commit** all of your changes to your local repository and the **Push** your changes to your personal GitHub remote repository.

4. Submit your assignment and attach the worksheet containing the test matrices for each of the three functions you created. In the comment section enter the url of your GitHub repository.