# COMP 322 Internet Systems Spring 2019 Assignment 10

Due Tuesday, May 7. 21 total points

**1** (6 pts.). In file **`prob1.js`**, define an new array method **`splitEqual()`** such that, if **`arr`** is an array, then **`arr.splitEqual(num)`** returns an array **`arrs`** of **`num`** arrays such that **`arrs[0]`** is a slice with the first few elements of **`arr`**, **`arrs[1]`** is a slice with the next few elements of **`arr`**, …, and **`arrs[num-1]`** is a slice with the last few elements of **`arr`**; every element of **`arr`** is in exactly one of these slices. As much as possible, the slices are of equal size. If **`arr.length`** isn't divisible without remainder by **`num`**, then some of the slices will have one more element than some of the others. Arrange it so that the longer slices are all near the beginning of **`arrs`**; note that the number of these slices is the length of **`arr`** modulo **`num`**.

As an example, if

```
var arr = [0,1,2,3,4,5,6,7,8,9,10,11,12,13];
```

then **`arr.splitEqual(3)`** returns the following (where the first two contained arrays have one more element than the third).

```
[[0,1,2,3,4], [5,6,7,8,9], [10,11,12,13]]
```

And **`arr.splitEqual(4)`** returns the following (where the first two contained arrays have one more element than the last two).

```
[[0,1,2,3], [4,5,6,7], [8,9,10], [11,12,13]
```

And **`arr.splitEqual(1)`** returns

```
[[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
```

Finally, **`arr.splitEqual(14)`** returns

```
[[0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]]
```

If **`num > arr.length`**, it isn't clear what it would mean to split **`arr`** into sub-arrays of nearly equal length, so, in that case, we throw an exception stating

**`this.length <`** ⟨*num*⟩

where ⟨*num*⟩ is the value of **`num`**.

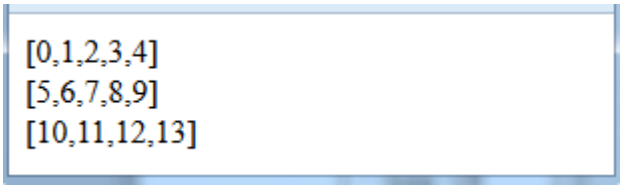
As a driver for testing your code, you are given (on the assignment page) the following HTML document, **`prob1.html`**.

```
<html>
<head>
  <meta charset="utf-8" />
  <title>Problem 2</title>
  <script type="text/JavaScript" src="prob2.js">
  </script>
  <script type="text/JavaScript" src="prob2Test.js">
  </script>
</head>
<body onload="start()">
  <div id="res"></div>
</body>
</html>
```

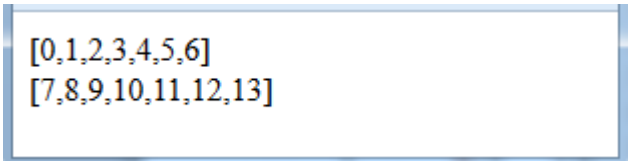
The following is a listing of the test file, **`prob1Test.js`**, which you are also given.

```
function start()
{
  var res = document.getElementById("res"),
      arr = [0,1,2,3,4,5,6,7,8,9,10,11,12,13],
      str="",
      a,
      i;
  try {
    a = arr.splitEqual(3)

    for ( i=0; i<a.length; i++ )
      str += "[" + a[i] + "]<br />\n";

    res.innerHTML = str;
  }
  catch(e) {
    alert(e.message)
  }
}
```

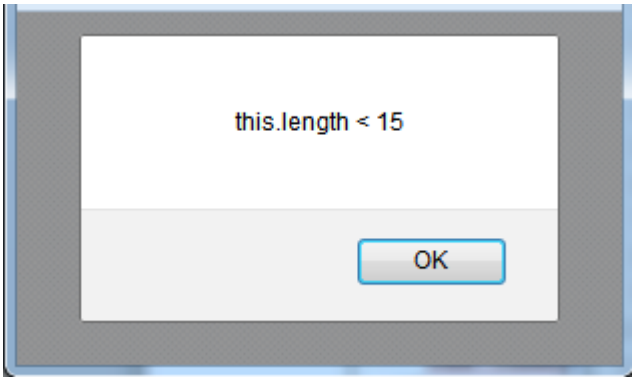
The rendering is as follows.




If we change the 3 in the call **`arr.splitEqual(3)`** in **`prob2Test.js`** to 2, we get




And if we change this 2 to, say, 15, we catch an exception.




Regarding the code in **`prob1.js`**, you assign to **`Array.prototype.splitEqual`** an anonymous function with one argument (say, **`num`**), specifying the number of slices in the array returned. Keep in mind that the array on which **`spliEqual()`** is invoked is denoted by **`this`** in the code in the anonymous function.

First check whether **`num > this.length`**. If so, construct an error,

```
new Error("this.length < " + num)
```

and throw it.

Next, create an empty array **arrArrs** for the slices; at the end, this array is returned. Suppose ***len*** is **this.length**. The number of elements in the shorter slices is the quotient of ***len*** by **num**, and the number of slices at the beginning with an extra element (whose length is the quotient plus 1) is the remainder of ***len*** by **num**. Since JavaScript does not have an operator for the quotient (integer division) of ***x*** by ***y***, we implement it as **Math.floor(x/y)**. We have variables **quotient** and **remainder**, set as follows.

```
quotient = Math.floor(this.length / num );
remainder = this.length % num;
```

We need two variables to delimit the part of **this** for the next slice. We maintain **low** so it is always the index of the first element in the next slice and **high** so that **high-1** is always the index of the last element. Initialization is

```
low = 0;
high = quotient;
```

We loop over **this** (the array **splitEqual()** is invoked on) **num** times, each time copying a slice to **arrArrs**:

```
for ( i=0; i<num; i++ ) {
```

Each time through this loop, we copy a slice of **this**, from **this[low]** through **this[high-1]**, to **arrArrs**, and we update **low** and **high** so that **low** has the old value of **high** and the new value of **high** is this old value plus **quotient**. At the beginning of an iteration, we decide whether the slice added in this iteration should contain an additional element. We use variable **remainder** to count down. If **remainder > 0**, increment **high** by 1 (to include the additional element) and decrement **remainder** by 1.

For this problem, submit your **prob1.js**.

**2** (3 pts.). You are given (on the assignment page) the following PHP file with a gap. Variable **`$arr`** is initialized to a 2D array where we can picture the rows as students and the columns as exams; the value in a cell is the score the student (row) received for the exam (column). Note, however, that there are different numbers of exams for different students, different exams are recorded for different students, and the order in which the exam-score pairs appear may differ from student to student. The missing code is a nested loop that sets **`$max`** to the largest exam score in the 2D array, **`$maxName`** to the name of the student with that score, and **`$maxExam`** to the name of the exam on which that student made that score. After the listing is a trace of the execution of the completed program.

Turn in this one file with the gap filled in. The easiest way to work out this program is to work with PHP on the command line (as discussed in the set of slides addressing this topic).

```
<?php
$arr = array(
         "Ed"  => array( "Exam 1" => 89, "Exam 2" => 76, "Exam 3" => 86 ),
         "Ken" => array( "Exam 2" => 86, "Exam 3" => 91, "Exam 4" => 81,
                         "Exam 5" => 78 ),
         "Sue" => array( "Exam 1" => 82, "Exam 3" => 71, "Exam 5" => 79 )
         );









echo "The highest score was $max, by $maxName on $maxExam.";
?>
```

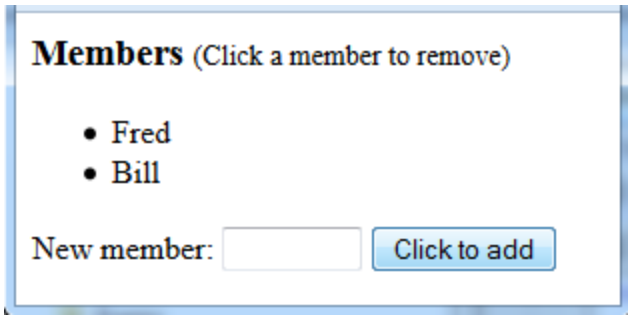
Execution:

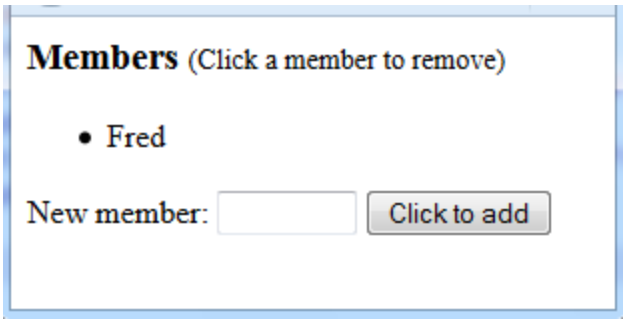
```
C:\SomeFolder>php prob1.php
The highest score was 91, by Ken on Exam 3.
```

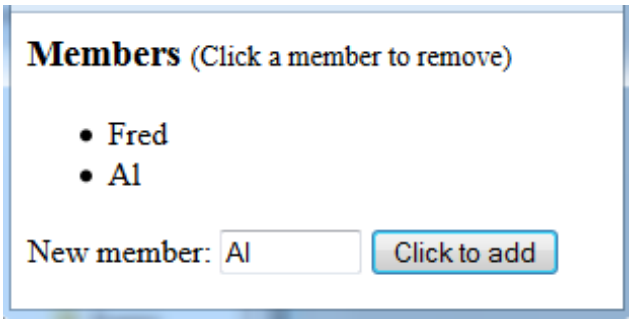
**3** (6 pts.). This problem is just like Problem 4 in Assignment 7 except that it uses jQuery. The following is a listing of HTML document **`prob3.html`**, which you can download from the assignment page.

```
<html>
<head>
  <meta charset="utf-8" />
  <title>Problem 3</title>
  <script type="text/javaScript" src="jquery-2.1.1.js">
  </script>
  <script type="text/JavaScript" src="prob3.js">
  </script>
</head>
<body>
  <h3>
    Members
    <span style="font-size:14; font-weight:normal">(Click a member to remove)</span>
  </h3>
  <ul id="mems">
    <li>Fred</li>
    <li>Bill</li>
  </ul>
  <p>
    New member: <input type="text" size="8" id="newm" />
    <input type="button" value="Click to add" id="bt" />
  </p>
</body>
</html>
```

Recall that this problem manipulates the DOM tree of a document by allowing the user to click on a list item to remove it and to add new list items using content in a textbox. The screenshot at right shows the initial rendering. The screenshot below left is the result of clicking on **Bill**, and the screenshot below right is the result of typing Al in the textbox and clicking the button **Click to add**. If **Al** (or **Fred**) were clicked here, it too would be removed.




Note that **`prob3.html`** (see the above listing) references a local copy of the jQuery distribution file, which you can download from the assignment page. It also references **`prob3.js`**, which you write.

File **prob3.js** should consist of a single anonymous function executed when the HTML has been loaded; it has the form

```
$('document').ready(function () {
  // You provide the code that goes here.
});
```

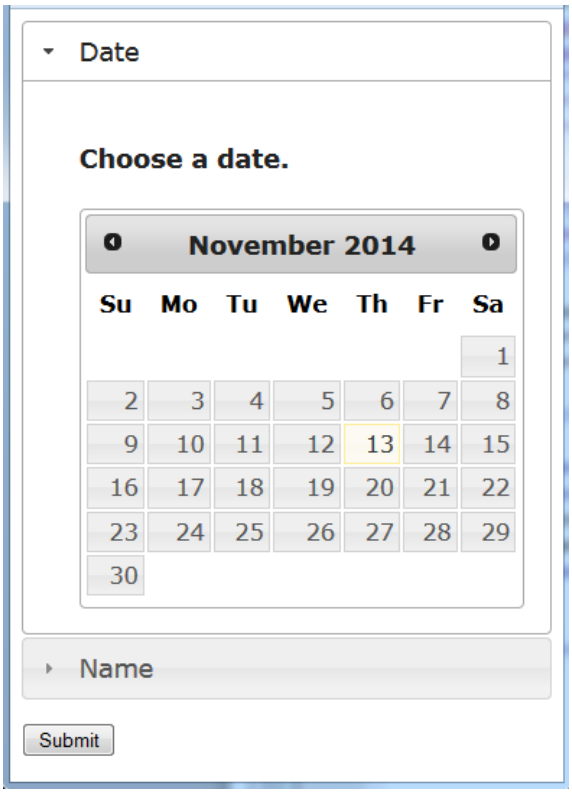
The body of the function consists of two jQuery statements. One statement makes as the click handler for **li** elements an anonymous function that removes the **li** element where the click occurs (think **this**). The other statement makes as the click handler for the button (with **id="bt"**) an anonymous function that creates an **li** element with the required click handler and appends it to the content of the **ul** element, after the **li** elements already there. The **li** element created is of the form

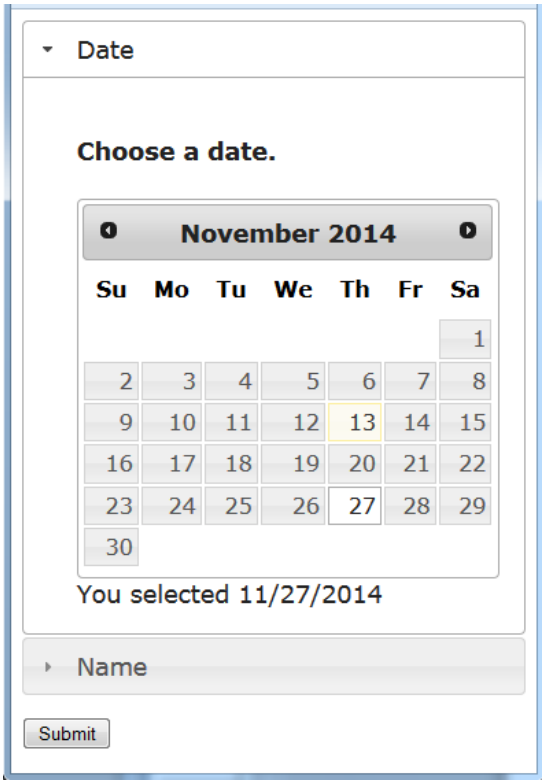
**<li>**⟨text⟩**</li>**

where ⟨text⟩ is the value (use method **val()**) of the textbox (with **id="newm"**). The click handler for this is the same as that for the **li** elements already present, viz., if removes the **li** element where the click occurs.

For this problem, submit your **prob3.js** file.

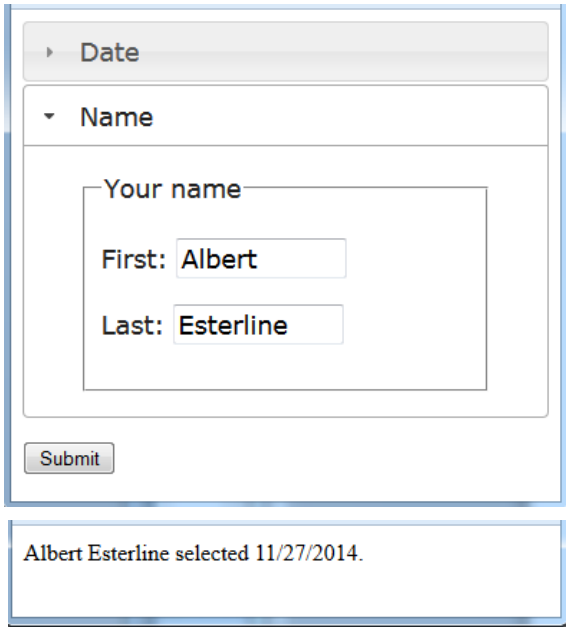
**4** (6 pts.). For this problem, you will use jQuery UI to produce an accordion display that is the content of a **form** element. The accordion has two pleats, one with a datepicker, and the other with a fieldset with two textboxes, for the user's first and last names. The screenshot below left shows the initial rendering. The screenshot to its right shows the rendering after the user clicked the date for Thanksgiving.




The screenshot at the top of the next page shows the rendering after the user clicked the title of the **Name** panel (expanding that pleat and collapsing the **Date** pleat) and entered values in the

textboxes for the first and last names. The screenshot below that shows the rendering of the response after the user clicked the **Submit** button with the two panels filled out as shown.




Write an HTML document **`prob4.html`** that renders as shown here. The **`form`** element specifies that, on submission, a GET request is sent to **`prob4.php`** (in the same folder). The textboxes in the **Name** panel have **`name="first"`** and **`name="last"`**, respectively. When the user clicks a date in the datepicker, the jQuery code (in file **`prob4.js`**) appends the date to the string **`"You selected "`** and makes the result the content of a **`div`** in this panel (see the second screenshot); it also inserts a hidden field after the submit button with **`name='date'`** and the selected date (e.g., **`'11/27/2014'`**) as its value.

The following is a listing of **`prob4.php`** (which you can download from the assignment page). Following this, we provide more detail on **`prob4.html`** and **`prob4.js`**.

```
<?php
  $first = $_GET['first'];
  $last  = $_GET['last'];
  $date  = $_GET['date'];
?>
<html>
<head>
  <meta charset="utf-8" />
  <title>Problem 4</title>
</head>
<body>
<?php
  echo "  <p>$first $last selected $date.</p>\n";
?>
</body>
</html>
```

At the top of the next page is a listing of **`prob4.html`** with a gap for the accordion code. You can download this file (still with the gap) from the assignment page. For the jQuery core and the jQuery UI JavaScript and CSS files, we are using the URLs for the CDN. Note that the submit button is inside a **`p`** element with **`id="sub"`**. The code filling the gap is a **`div`** element with **`id="accordion"`**. In the first panel, include two **`div`** elements: one, with **`id="datep"`**, to hold the datepicker, and the other, with **`id="selected"`**, to be given the **`"You selected`** …**`"`** text when the user picks a date.

```
<html>
<head>
  <meta charset="utf-8" />
  <title>Problem 4</title>
  <link rel="stylesheet"
    href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css" />
  <script type="text/JavaScript" src="http://code.jquery.com/jquery-1.10.2.js">
  </script>
  <script type="text/JavaScript" src="http://code.jquery.com/ui/1.11.2/jquery-ui.js">
  </script>
  <script type="text/JavaScript" src="prob4.js">
  </script>
</head>
<body>
  <form action="prob4.php" method="get">
    <p id="sub"><input type="submit" value="Submit" /></p>
  </form>
</body>
</html>
```

File **prob4.js** again should consist of a single anonymous function executed when the HTML has been loaded; it has the form

```
$('document').ready(function () {
  // You provide the code that goes here.
});
```

The body of the function consists of two jQuery statements. One statement makes the **div** with **id="accordion"** hold the accordion. So that the panels are large enough to hold their content, we want to invoke the **accordion()** method with a **heightStyle** parameter whose value is **"content"**. Recall that parameters to these methods are actually key-value pairs in an object literal, so our call should be

```
accordion({ heightStyle: "content" })
```

The second jQuery statement in the body of the anonymous function makes the **div** element with **id="datep"** hold the datepicker. We pass **datepicker()** one parameter, **onSelect**, with an anonymous function as its value, so the call has the form

**`datepicker( { onSelect:`** ⟨*function*⟩**`} )`**

where ⟨*function*⟩ is the function called when a selection is made (by clicking) in the datepicker. Recall that, when the system calls this function, it passes it (as first argument) the date in string form, **`"mm/dd/yyy"`**, and (as second argument, one that does not concern us here) an object corresponding to the datepicker widget. The body of this function should in turn consist of two jQuery statements. The first statement sets the text of the **`div`** with **`id="selected"`** to a string of the form **`"You selected`** ⟨*date text*⟩**`"`**, where ⟨*date text*⟩ is the string that is the first parameter of this anonymous function.

The second statement in the body of this function creates a hidden field of the form

**`<input type='hidden' name= 'date' value='`**⟨*date text*⟩**`' />`**

(where ⟨*date text*⟩ is as above) and inserts it just after (and outside) the **`p`** element with **`id="sub"`**. You must be careful with quotation marks when constructing the above **`input`** element. To keep you from getting snagged on this issue, I here give you my code for the string from which this element is formed:

```
"<input type='hidden' name= 'date' value='" + dateText + "' />"
```

Here **`dateText`** is the first parameter of the anonymous function (what above we denoted with ⟨*date text*⟩).

For this problem, submit files **`prob4.html`** and **`prob4.js`**.