Computer Science Correspondence School

discere domi

Lecture 11

CISC 131
Introduction to Programming and Problem Solving
Spring 2020
Arrays and Dates

Due: **Friday, April 17, 2020, at start of class**
Points: **20**

Solutions to Problems from Lecture 10

Here are some sample solutions to the array exercises you did last time. Your solutions may differ from mine, so study mine to see other ways to do the functions. There are almost always several algorithms that can be used to solve a problem.

Problem 1

Here is a function that returns the count of the occurrences of *countMe* in the *String* source.

```
function countInString ( source, countMe )
{
 var i;
 var count;

 i = 0;
 count = 0;
 while ( i < source.length )
 {
  if ( source.charAt(i) === countMe ) { count = count + 1; }
  i = i + 1;
 }

 return count;
}
```

Write the equivalent function that has an array parameter and returns the number of elements in the array that have the value *countMe.*

```
function countInArray ( array, countMe )
{
 var i;
 var count;

 i = 0;
 count = 0;
 while ( i < array.length )
 {
  if ( array [ i ] === countMe ) { count = count + 1; }
  i = i + 1;
 }
 return count;
}
```

Problem 2

Here is a function that is passed a *String*. It returns a *String* that contains all the characters of the parameter *String* but changes all *find* characters into *replace* characters. For example, if the function were called this way:

```
changeString("abcde","b","Z")
```

it would return *aZcdeZ*

```
function changeString ( source, find, replace )
{
 var ch;
 var i;
 var result;

 result = "";
 i = 0;
 while ( i < source.length )
 {
  ch = source.charAt ( i );
  if ( ch === find ) ch = replace;
  result = result + ch;
  i = i + 1;
 }

 return result;
}
```

Now write a function that is passed an array. It returns a newly created array that contains all the values of the parameter array but changes all *find* values into *replace* values.

```
function changeString ( source, find, replace )
{
 // return a String in which all the occurrences of
 // character one are replaced by character two
 var ch;
 var i;
 var result;

 result = "";
 i = 0;
 while ( i < source.length )
 {
  ch = source.charAt ( i );
  if ( ch === find ) ch = replace;
  result = result + ch;
  i = i + 1;
 }

 return result;
}
```

Problem 3

Here is a function that is passed a *String*. It returns a *String* that contains all the characters of the parameter String but swaps character number *i* and character number *j*. For example, if the function were called this way:

```
swapCharacters("ABCDEF",3,2)
```

it would return *ABDCEF*

```
function swapCharacters ( source, i, j )
{
 var hold;
 var result;

 if ( i === j )
 {
  result = source;
 }
 else
 {
  hold = Math.min ( i, j );
  j    = Math.max ( i, j );
  i    = hold;

  result = source.substring ( 0, i )
         + source.charAt ( j )
         + source.substring ( i+1, j )
         + source.charAt ( i )
         + source.substring(j+1);
  }

 return result;
}
```

Now write a function that is passed an array. It returns a newly created array that contains all the values of the parameter array but with the values of elements *i* and *j* swapped. The values in the parameter array must not be changed in the function.

```
function swapElements ( array, i, j )
{
 var hold;
 var k;
 var result;

 result = new Array ( array.length );
 k = 0;
 while ( k < result.length )
 {
  result [ k ] = array [ k ] ;
  k = k + 1;
 }

 hold = result [ i];
 result [ i ] = result [ j ] ;
 result [ j ] = hold;

 return result;
}
```

Start of Today's Lecture

As you know, the HTML file is associated with a Javascript file by using the *script* tag in the HTML document. The *src* parameter of the *script* tag identifies the Javascript file that will be used. Here is an example that associates the file *MyFile.js* with the HTML document.

```
<script src="MyFile.js" type="text/javascript"></script>
```

If the file cannot be found, the tag is ignored. While you have done this for every Javascript assignment, you may not know that HTML allows multiple *script* tags. This means that you can have multiple Javascript files associated with one HTML document. All the script tags should be grouped together and be careful. Only one of the Javascript files should contain the *window.onload* function. For example:

```
<script src="Animation.js" type="text/javascript"></script>
<script src="Utilities.js" type="text/javascript"></script>
<script src="LoanFunctions.js" type="text/javascript"></script>
```

Using multiple script tags can help you organize your Javascript functions into "libraries". When writing a new program, you simply use a *script* tag to import the "library" and now your new program does not need to have lots of functions that it needs to use but you wrote long ago. Be careful, though:

- Each function definition should occur only once and in only one *js* file
- All the *js* files must be in the same folder. This means you will have to copy and paste them into a new folder for each new project that you do.

Step 1:

Create a *Utilities.js* file that contains these function headers and their associated bodies.

```
function getRandomInteger ( upperLimit )
function getRandomRGB()
function countElementsWithIdPrefixOf ( idPrefix )
function createHTMLElement(elementType, id, classInfo, content)
```

To make these functions available, you would use this *script* tag in the HTML file

```
<script src="Utilities.js" type="text/javascript"></script>
```

When you decide to create libraries like this, you must include documentation at the start of the *js* library file that includes your name and a manifest of the functions contained in the library. For each function, the manifest would include the function header, what the function does, and how to use it. It's important to do this. Memory fades and you will forget what your functions do. Having some documentation will help jog your memory. Here is a partial example:

```
/*
Author: Anna Nutherthing
        CISC 131, Spring 2020

function countElementsWithIdPrefixOf ( idPrefix )
     returns the number of HTML elements that have the parameter as the
     prefix of an HTML element id and an integer suffix. The suffix starts
     at zero. Suffixes must be sequential. E.g., box0, box1, box2 …

function createHTMLElement(elementType, id, classInfo, content)
etc.
.
.
.
*/
```

This will also make it much easier for you to find what's in the file when you want to use it later.

Step 2:
Since we have developed a number of functions that processes dates, it will be useful to create a *Dates.js* file that contains these functions. You will be writing more date functions in this assignment and changing some of the existing ones to use arrays rather than *Strings*, so, for now, only place the following functions in the *Dates.js* library.

```
getDay   ( yyyymmdd )
getMonth ( yyyymmdd )
getYear  ( yyyymmdd )

setDay   ( yyyymmdd )
setMonth ( yyyymmdd )
setYear  ( yyyymmdd )

isLeapYear    ( year )
isValidYear   ( year )

getDaysInYear ( year )
```

Step 3:
Write and thoroughly test the following functions. Place these functions in a Javascript file that has this name:

YourLastName-Dates20200417.js

where *YourLastName* is your last name. The HTML file that you create should have two script tags:

```
<script src="Dates.js" type="text/javascript"></script>
<script src="YourLastName-Dates20200417.js" type="text/javascript"></script>
```

Here are the functions that must be included in the *YourLastName*-Dates20200417.js file: Make sure you test them thoroughly. Only the last two functions listed use arrays. The others do not use arrays but may use some of the functions listed above in Step 2. Of course, you can use as many Javascript statements as you want to when implementing the non-array functions but each of them can be implemented using a single Javascript statement.

`function getEarlierDate ( yyyymmdd1, yyyymmdd2 )`
This function is passed two dates in the *yymmdd* format and returns the one that is earlier in time. If both dates are the same, either one can be returned because they are the same.

`function getLaterDate  ( yyyymmdd1, yyyymmdd2`

This function is passed two dates in the *yymmdd* format and returns the one that is later in time. If both dates are the same, either one can be returned because they are the same.

`function sameDate ( yyyymmdd1, yyyymmdd2`

This function returns Boolean *true* if the parameters are the same date and *false* if they are not the same date.

`function sameYear  ( yyyymmdd1, yyyymmdd2`

This function returns Boolean *true* if the two parameter dates have the same year value and *false* if they do not have the same year value.

`function sameMonth ( yyyymmdd1, yyyymmdd2 )`

This function returns Boolean *true* if the two parameter dates have the same month value and *false* if they do not have the same month value.

`function sameDay  ( yyyymmdd1, yyyymmdd2 )`

This function returns Boolean *true* if the two parameter dates have the same day value and *false* if they do not have the same day value.

`function getDayName ( dayNumber )`

This function is passed an integer value. If the value is between one and seven, it returns a *String* containing the name of the week associated with that value. *Sunday* is associated with value one, *Monday* is associated with value two, and so on, with *Saturday* being associated with value seven. If the value is not between one and seven, the function returns a zero length *String*.

You must use an array to store and access the names of the days. This function can be implemented without any *if* statements but you may use at most one *if* statement.

`function getMonthName ( monthNumber )`

This function is passed an integer value. If the value is between one and twelve, it returns a *String* containing the name of the month associated with that value. *January* is associated with value one, *February* is associated with value two, and so on, with *December* being associated with value twelve. If the value is not between one and twelve, the function returns a zero length *String*.

You must use an array to store and access the names of the months. This function can be implemented without any *if* statements but you may use at most one *if* statement.

What You Must Turn In

- Send me an email with this subject line:

  CISC131-Dates-20200417

- Attach the ***YourLastName*-Dates20200417.js** to the email message and send it to me.
- Do NOT attach the *Dates.js* file.