
*Computer Science Correspondence School*

*discere domi*

**Lecture 9**

# CISC 131
# Introduction to Programming and Problem Solving
# Spring 2020
# Array Problem Set

**Due: Wednesday, April 8, 2020**
**Points: none**

## Solutions to Problems from Lecture 8

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

### Exercise A

Create the function with the following header:

```
function copy ( array )
```

The function is passed an array whose contents are unknown – they could be *Strings*, they could be numbers, they could be anything. The function returns an array that is the same size and contains the same contents as the passed array. Just returning a reference to the passed array will not work. You must duplicate the array. This means you must create a new array and copy each of the elements from the passed array into the new array and then return the variable that contains a reference to this new array.

```
function copy ( array )
{
 var i;
 var result;

 result = new Array ( array.length );

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

 return result;
}
```

### Exercise B

Create the function with the following header:

```
function letterCount ( someString )
```

The function is passed a *String* and returns an array. The function returns an array of length two. The first component of the returned array contains the number of vowels (in either upper or lower case) in the parameter. The second component contains the number of non-vowels in the parameter.

```
function letterCount ( someString )
{
 var ch;
 var i;
 var result;
 var vowels;

 vowels = "aeiouyAEIOUY";
 result = new Array ( 2 );

 //  initialzze the counters
 result [ 0 ] = 0;
 result [ 1 ] = 0;

 i = 0;
 while ( i < someString.length )
 {
  ch = someString.charAt ( i );

  if ( vowels.indexOf ( ch ) >= 0 )
   result [ 0 ] = result [ 0 ] + 1;
   else
   result [ 1 ] = result [ 1 ] + 1;

  i = i + 1;
 }

 return result;
}
```

**Exercise C**

Create the function with the following header:

```
function sum ( array )
```

The function is passed an array of numbers. It does not return an array. It returns a number which is the sum of the values in the parameter array

```
function sum ( array )
{
 var i;
 var result;

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

 return result;
}
```

**Exercise D**

Create the function with the following header:

```
function arrayToString ( array )
```

The function is passed an array of unknown contents. It does not return an array. It returns a *String* which is the result of concatenating together each component of the parameter array. Remember that the parameter array might not contain *Strings*, it might contain numbers and that your function cannot change the contents of any component of the parameter array.

```
function arrayToString ( array )
{
 var i;
 var result;

 result = "";
 i = 0;
 while ( i < array.length )
 {
  result = result + array [ i ];
  i = i + 1;
 }

 return result;
}
```

**Exercise E**

Create the function with the following header:

```
function getStringsOfLength( anArray, numberOfCharacters )
```

The function is passed two parameters: an array containing *Strings* (*anArray*) and an integer (*numberOfCharacters*). The function returns an array whose contents are the *Strings* in the parameter array that have the same number of characters as the value of the parameter integer. For example, if the parameter integer had the value seven, the returned array would contain copies of all the *Strings* in the parameter array that contained exactly seven characters.

```
function getStringsOfLength( anArray, numberOfCharacters )
{
 var count;
 var i;
 var result;

 // Step 1
 // Count the number of Strings in the array that have the correct
length
 count = 0;
 i     = 0;
 while ( i < anArray.length )
 {
  if ( anArray[i].length === numberOfCharacters )
  {
   count = count + 1;
  }
  i = i + 1;
 }

 // Step 2
 // Create the array
 result = new Array ( count );

 // Step 3
 // Update the array elements
 count = 0;
 i     = 0;
 while ( i < anArray.length )
 {
  if ( anArray[i].length === numberOfCharacters )
  {
   result [ count ] = anArray [ i ];
   count = count + 1;
  }
  i = i + 1;
 }

 return result;
}
```

## Start of Today's Problem Set

Implement and test each of the following functions. Test thoroughly. It will be helpful to display the contents of the array during your testing process. The *arrayDisplay* function you wrote previously is a good way to do this. Unless explicity told to do so, the functions that you write must not change the value of any element of the formal parameter array.

**Exercise A**

Create the function with the following header:

```
function addOrConcatenate ( array )
```

The formal parameter array either contains all *Strings* or all numbers and it contains at least one component (*i.e*., its *length* property is greater than zero). If the array contains all numbers, the function returns their sum. If the array contains all *Strings*, the function returns all of the array elements concatenated together into one *String*. You may not use any Javascript that has not been introduced in class. For example, you may not use the *instanceof* operator. The solution is easy but determining what the solution is might take some time. Take some time and think about it. Test the function using an array of numbers and an array of *Strings*.

**Exercise B**

Create the function with the following header:

```
function deleteElementZero ( array )
```

The function is passed an array and returns a newly created array. The array that is returned by the function has all the elements of the parameter array with the exception of element zero. For example, the parameter array contained these elements: *5, 9, 3 4* then the array returned by the function would contain only these elements: *9, 3, 4*. So, the *length* of the array returned by the function is one less than the *length* of the parameter array. If the parameter array *length* is zero, then the function should return a newly created array whose *length* is also zero.

**Exercise C**

Create the function with the following header:

```
function deleteLastElement ( array )
```

The function is passed an array and returns a newly created array. The array that is returned by the function has all the elements of the parameter array with the exception of the last element of the parameter array. For example, the parameter array contained these elements: *5, 9, 3 4* then the array returned by the function would contain only these elements: 5, *9, 3*. So, the *length* of the array returned by the function is one less than the *length* of the parameter array. If the parameter array *length* is zero, then the function should return a newly created array whose *length* is also zero.

**Exercise D**

Create the function with the following header:

```
function locationInArray ( array, findMe )
```

The function is passed an array and a value. The function returns an integer. The return value is -1 if the value of *findMe* is not in the array. If the value of *findMe* is in the array, the function returns the element number of the array element whose value is equal to the value of *findMe*. If the value of *findMe* occurs more than one time in the array, the function should return the location of the match that is nearest to the beginning of the array.

**Exercise E**

Create the function with the following header:

```
function deleteKthElement ( array, k )
```

The function is passed an array and an integer *k*. The function returns an array. If the value of *k* is a valid subscript value for the parameter array (>=0 and <*array.length*), the function returns a new array that contains all the elements of the parameter array except element number *k*. That means that the array returned by the function will have one fewer elements than the parameter array. If the value of *k* is not a valid subscript value for the parameter array, the function returns a new array that contains all the elements of the parameter array.

**Exercise F**

Create the function with the following header:

```
function deleteAllOccurrences ( array, findMe )
```

The function is passed an array and *findMe*, a value of any kind (*String*, number, etc.). The function returns an array. The array that is returned has all the values of the parameter array after all the *findMe* values have been deleted. If *findMe* is not in the parameter array, then the return value is a copy of the parameter array.