CPSC 1045: The <select> HTML tag and Arrays

The <select> Tag

To obtain input from the user, so far we have seen the <input> tag. In particular we use the <input type="text"> to use a text field. One problem with the text field is that the user can enter any value. If want to only give the user a few options for the values they can enter, we can use the <select> tag. The select tag has <option> tags inside it to define the options that user can pick. If we want to give the user the option to choose the value "salt" we would write:

```
<option value = "salt">Salt</salt>
```

The text between the open and close tags is what is displayed to the user. The string after `value=` is the value that we will see in JavaScript, in this case the string "salt". The two values do not need to be the same.

To define a select box with three options we can write the following:

```
<select id="ingredientInput">
   <option value="salt">Salt</option>
   <option value="pepper">Pepper</option>
   <option value="allspice">All Spice</option>
<select>
```

This is create a selection box that gives the user three option.

In JavaScript to access the value the user entered, we will get a reference to the element and access the value property of the element.

```
var element = document.getElementById("ingredientInput");
var S1 = "The user entered " + element.value;
```

Arrays:

Arrays in JavaScript can be considered a list, or a set of related variables. JavaScript arrays supports mixed types, so they are more flexible than arrays in other languages.

If you ever had the need to create multiple variables with same name, such as `var1,var2,var3, var4` because they are variables that represent different instances of the same thing, then it means you probably should be using an array. Alternatively, if you ever needed a list, such as a list of names, list of ingredients, list of points etc...that also when arrays can be useful.

Creating an array

For arrays, we will use square brackets alot.. To define an empty array we simply write ,`[ ]`.

The line of code below creates an empty array and assigns it to the variable `arrayExample1`.

var arrayExample1= [];

To define an array with values in it, we use the square brackets follow by a comma separated list of those values.

```
var arrayExample=["dog", "cat", "bird", "fish"];
```

The above example creates a list with 4 elements.

Index	**value**
0	“dog”
1	“cat”
2	“bird”
3	“fish”

Each element has an index, and indexes in arrays starts from 0. We can access the stored values by their index.

We start with the name of the array, followed by the square bracket. Inside the square bracket, we place the index. So, earlier we used the square brackets to create the array, in which it was **not** preceded by the array name. Now we will use the square bracket as the element access operator.

`arrayExample[2]` will have the value “bird” because that is the string at index 2.

Similary we can modify the elements of the array using the same pattern.
Continuing from the previous example
`arrayExample[2] = “dinosaur”`

Index	**value**
0	“dog”
1	“cat”
2	“dinosaur”
3	“fish”

The previous example then changed the element at index 2 to "dinosaur" from "bird".

Creating List of mixed types:

A feature of JavaScript is that list can contain a mixed of types.

```
var ArrayExample3 = [ 12, 89, "Dog", "cat", 9];
```

The above example shows that we can mix types in a JavaScript array. We are mixing strings and numbers.

Index	value
0	12
1	89
2	"Dog"
3	"cat"
4	9

Adding elements to the array

Array sizes are not fixed in JavaScript, that is you can add elements to an array and remove elements from an array.

The push method

Arrays are objects, which means they have methods. The **push** method is used to add elements to the end of the array.

```
arrayExample3.push(45);
```

will add 45 to the end of ArrayEcample3

Index	value
0	12
1	89
2	"Dog"
3	"cat"
4	9
5	**45**

Assigning to an missing index

A second way to add elements to an array is just to assign some value to t a missing index. The index will be created and assigned a value. This leads to the possibility of having a set of indexes that are discontinuous.

Continueing with `arrayExample3` there is no index larger than 5. What if we assigned values to index 6 and index 8, as shown below?

```
arrayExample3[6] = "New element";
arrayExample3[8] ="What happened to Seven";
```

Index	value
0	12
1	89
2	"Dog"
3	"cat"
4	9
5	45
6	"New element"
8	"What happened to Seven"

So there are two interesting things. We created two new elements, one with index 6 and one with index 8. The index 7 is missing, and this is fine in JavaScript.

It is not recommended that you use this feature, however if you make a programming mistake you may accidentally use this feature, especially if you get your **for..loop** indexes wrong.

Removing Elements from an array

To remove elements you can use the pop method. It will remove the last element of the array, and return the value.

```
var elementValue = arrayExampy3.pop();
```

The statement arrayExampy3.pop(); will return "What happened to Seven", and remove that element from the array.

Index	value
0	12
1	89
2	"Dog"
3	"cat"
4	9
5	45
6	"New element"

unshift/shift

Arrays also has the `unshift` and `shift` methods. The `unshift` method is used to add an element to the front of the list. This will cause the indexes of all the other elements to go up by one. The `shift` method is used to remove the first element

from the list. This will cause the indexes of all the other elements to go down by one.

Copying an array

To copy an array of numbers or strings, we can use the slice() method on the array.

```
var newArray  = oldArray.slice();
```

And changes to newArray will on affect oldArray, and any changes to oldArray will not affect newArray.

This technique will not work for arrays containing objects, or other arrays.

Processing arrays:

A common technique for processing or manipulating arrays is to use a **for loop**. This technique will only work if you indexes are continuous, and ideally starts from 0. If we wanted to multiple all the array elements by 2, we will need two things.

1. The .length property that tells us how many elements are in the array
2. A for loop to process the array one element at a time.

The example below multiplies all elements by 2.

```
for(var i = 0; i < arr.length; i += 1){
    arr[i] = arr[i]  * 2;
}
```

Passing an array in as a parameter

Arrays can be passed in as parameters like any other type in JavaScript. Because JavaScript is weakly typed, there are no restrictions on what can be passed as a parameter.

Example:

```
function useArray(ctx, inputArray){
    "use strict";
   // ctx is our usual drawing context
   // inputArray is a array of sizes
   for(var i = 0; i < inputArray.length; i =  i +1){
      ctx.fillRect(i*10, i*1, inputArray[i], inputArray[i]);
    }
}
```

calling our function:

```
var array1 = [10,20,10,10];
```

```
useArray(ctx, array1);
```

The above example will draw a set of squares, whose sizes are specified by the array.

Returning an array

We can also return an array like we do any value. We simply set the array as the return expression.

```
function createSquares(N){
   "use strict";
   var arrayOfSquares = [];
   for(var i = 1; i <= N; i = i +1){
      arrayOfSquares.push(i*i);
    }
    return arrayOfSquares;
}
```

The above example returns an array of the first N square values.

Creating a 2-D array

2-D arrays are simply arrays of arrays. We go back to understanding that in JavaScript everything is an expression. To create an 2-D array we can write:

arr2D = [[1,2,3], [4,5,6], [7,8,9]];

arr2D[0] is an expression. In this case, it will evaluate to [1,2,3].

If we wanted to access the number 3 we would write:
arr2D[0][2] and this expression will evaluate to 3.

The square brackets is officially known as the "Computed Member Access" operator. And has a precedence of 18, the second highest of all the operators. More informally it's the "index operator" or the "element selection operator". It operates on any arrays to the left of it, and ahs the index between it.

So if we have arrays of arrays, when we want to access a value, like 3 we need to use two "selection operator". The first operator selects the row, of the array we want, and the second selects the element within that array, which leads to the **double bracket** notation.