Reference Types: Part 1

.NET Cohort

SOFTWARE GUILD

Lesson Goals

1. Learn how Objects are created and used
2. Learn about the Array type

The Object Type

Like in other languages, objects exist for the purpose of encapsulating data to be stored or passed around your application.

There are two ways to create an object:

1. Using the new Object() constructor
2. Using *object literal* notation

Most current-day developers use object literal notation.

```
// object constructor
    var person = new Object();
    person.name = "Nicholas";
    person.age = 29;

// object literal
    var person2 = {
        name : "Nicholas",
        age : 29
    };


function displayInfo(args) {
    var output = "";
    output += "Name: " + args.name + "\n";
    output += "Age: " + args.age + "\n";
    alert(output);
}

displayInfo({
    name: "Nicholas",
    age: 29
});

displayInfo({
    name: "Greg"
});
```

Bracket Notation

Less often used for object properties is bracket notation. This allows you to declare properties with otherwise-invalid characters or reserved words. To do this, put the property name in quotes and then access it with brackets.

```
var person ={
    "First Name": "Nicholas",
    age: 29
};

console.log(person["First Name"]);
```

That’s Pretty Much it for Objects

There’s not much concern for defining data types, so spinning up a bunch of properties on a JavaScript object is trivial.

Just be careful with your casing and spelling.

Arrays

Unlike in other classes, Arrays in JavaScript can hold any type of data in each slot (element). So it’s possible to have a number in the first position, string in the second, and object in the third (though a bad design!).

JavaScript arrays are also dynamically sized, which means they automatically grow if you add more data to them.

Declaring Arrays

We can declare arrays in a few ways:

1. Using the empty constructor
2. Using the constructor and passing it a length
3. Using the constructor and passing in items for the array
4. Using the array literal notation

```
// 1. Array Constructor
var arr = new Array();

// 2. Array Constructor with length
var arr2 = new Array(20);

// 3. Array Constructor with values
var arr3 = new Array("red", "blue", "green");

// 4. Array literals

var arr4 = ["red", "blue", "green"];
var arr5 = []; // empty array

var colors = ["red", "blue", "green"]; //define an array of strings
alert(colors[0]); //display the first item
colors[2] = "black"; //change the third item
colors[3] = "brown"; //add a fourth item
```

To access elements of the array, we use indexes like in C#. Note that even if we define an array with 3 elements, we can use the fourth without any difficulty.

Array Length

Length is read-only in C#, but in JavaScript we can write to it and if we set it lower than the current size, the extra items get discarded.

Length-1 is the last position of the array.

Array elements with no data are *undefined.*

```
var colors = ["red", "blue", "green"]; //creates an array with three strings
var names = []; //creates an empty array

alert(colors.length); //3
alert(names.length); //0

var colors = ["red", "blue", "green"]; //creates an array with three strings
colors.length = 2;
alert(colors[2]); //undefined
```

Checking for Arrays

There is a handy function on the Array class called IsArray which returns true if a given variable contains an array.

```
var colors = ["red", "blue", "green"];
if (Array.isArray(colors)) {
    //do something on the array
}
```

Converting Arrays

toString() and valueOf() return a comma-separated string of the array values. (People sometimes erroneously expect C# to do this, too.)

```
var colors = ["red", "blue", "green"];
alert(colors.toString()); //red,blue,green
alert(colors.valueOf()); //red,blue,green
alert(colors); //red,blue,green
```

We can override the commas by using the .join() method.

```
var colors = ["red", "green", "blue"];
alert(colors.join(",")); //red,green,blue
alert(colors.join("||")); //red||green||blue
```

Arrays Can Act Like Stacks

Arrays out of the box have push() and pop() methods. push() adds an item to the back of the array and returns the new count of items, pop returns the last item in the array and removes it.

```
var colors = new Array(); //create an array
var count = colors.push("red", "green"); //push two items
alert(count); //2

count = colors.push("black"); //push another item on
alert(count); //3

var item = colors.pop(); //get the last item
alert(item); //"black"
alert(colors.length); //2
```

And Also Queues…

If we want to take off the first element instead of the last, we use shift() instead of pop().

```
var colors = new Array(); //create an array
var count = colors.push("red", "green"); //push two items
alert(count); //2

count = colors.push("black"); //push another item on
alert(count); //3

var item = colors.shift(); //get the last item
alert(item); //"red"
alert(colors.length); //2
```

Adding Items to the Front?

Sure, why not? We can use unshift() to add items to the front.

```
var colors = new Array(); //create an array
var count = colors.unshift("red", "green"); //push two items
alert(count); //2

count = colors.unshift("black"); //push another item on
alert(count); //3

var item = colors.pop(); //get the last item
alert(item); //"green"
alert(colors.length); //2
```

Reordering Arrays

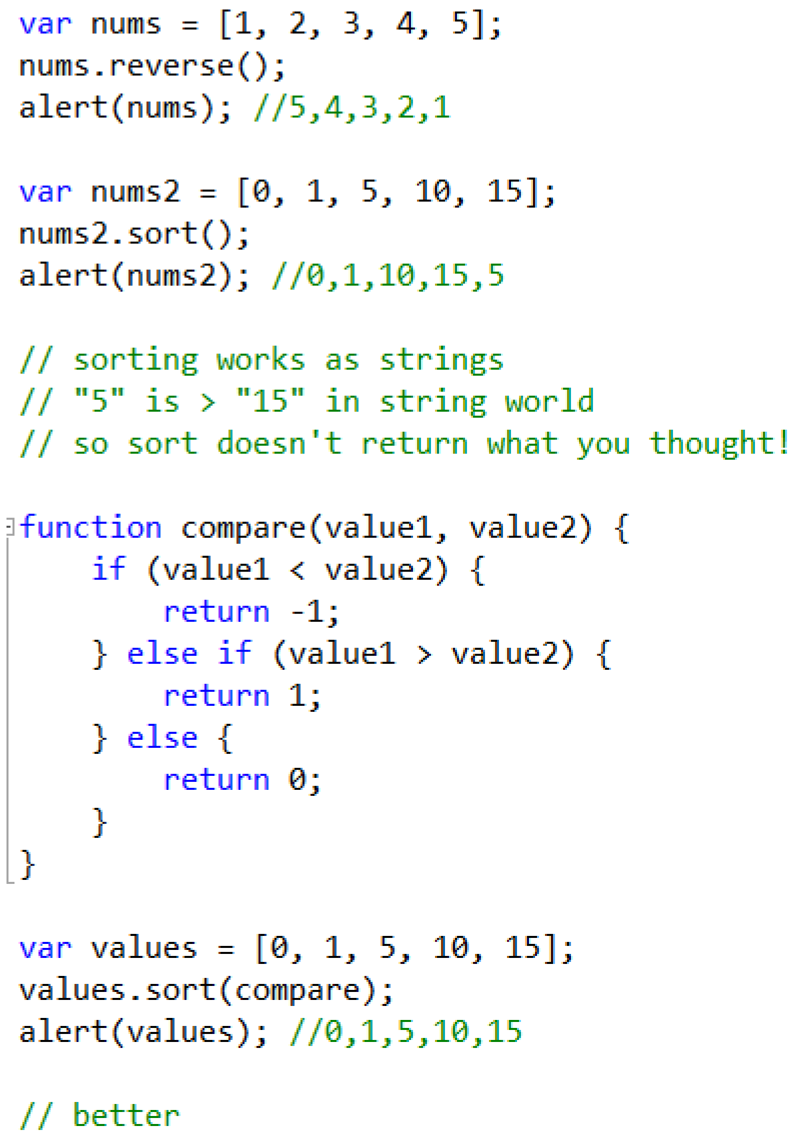
We can reorder arrays using sort() and reverse().

Optionally, we can pass a sort function into the sort() command to do something custom.

Like the IComparable in C#, the compare function returns negative, positive, or 0 depending on whether value1 is greater, less than, or equal to value2.

We could simplify a number compare with the following code:

```
function compare(n1, n2)
{
    return n1-n2;
}
```



```
var nums = [1, 2, 3, 4, 5];
nums.reverse();
alert(nums); //5,4,3,2,1

var nums2 = [0, 1, 5, 10, 15];
nums2.sort();
alert(nums2); //0,1,10,15,5

// sorting works as strings
// "5" is > "15" in string world
// so sort doesn't return what you thought!

function compare(value1, value2) {
    if (value1 < value2) {
        return -1;
    } else if (value1 > value2) {
        return 1;
    } else {
        return 0;
    }
}

var values = [0, 1, 5, 10, 15];
values.sort(compare);
alert(values); //0,1,5,10,15

// better
```

SOFTWAREGUILD

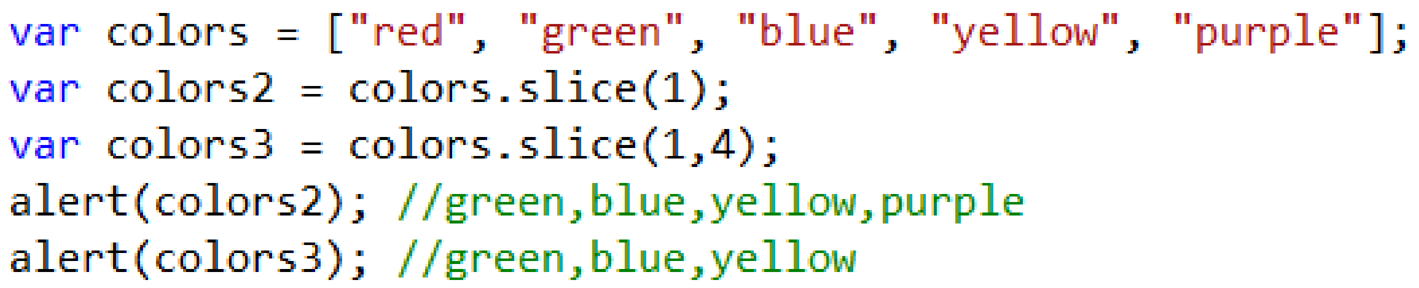
Adding Many Items

The concat() method allows us to add a list of items (or another array) to an existing array.

```
var colors = ["red", "green", "blue"];
var colors2 = colors.concat("yellow", ["black", "brown"]);
alert(colors); //red,green,blue
alert(colors2); //red,green,blue,yellow,black,brown
```

Getting a Subset of Items

The slice() method allows you to specify a starting element and optional ending element. If you omit the ending element, it will select all items after the start; otherwise, it will select only the items between the bounds, non-inclusive of the ending index.



```
var colors = ["red", "green", "blue", "yellow", "purple"];
var colors2 = colors.slice(1);
var colors3 = colors.slice(1,4);
alert(colors2); //green,blue,yellow,purple
alert(colors3); //green,blue,yellow
```

Manipulating Arrays

The splice() method is very powerful. It allows for insertion, deletion, or replacement of array elements.

If you only specify two positions, the items will be deleted.

If you specify starting position, the number of items to delete, and then some data it will delete the items at the starting position and insert the data in that spot. The deleted and added items don’t need to be the same size.

Conceptually, a delete and insert is a replace.

Manipulating with splice()

```
var colors = ["red", "green", "blue"];
var removed = colors.splice(0,1); //remove the first item
alert(colors); //green,blue
alert(removed); //red - one item array

removed = colors.splice(1, 0, "yellow", "orange"); //insert two items at position 1
alert(colors); //green,yellow,orange,blue
alert(removed); //empty array

removed = colors.splice(1, 1, "red", "purple"); //insert two values, remove one
alert(colors); //green,red,purple,orange,blue
alert(removed); //yellow - one item array
```

Conclusion

As you can see, JavaScript, being rooted in C, has a lot in common with C# and many of the common structures we’re already used to, like statements and loops, translate over.

All languages have these concepts, so learning new languages after you master one is not all that difficult.

SOFTWARE GUILD