**JavaScript Basics #3: Arrays and Objects**

Previously, we discussed some data types that are built into JavaScript, like strings, numbers and boolean values. In this article, we are going to talk about two new data types that allow us to group all of these values together to create more complex structures.

Table of Contents

* [Arrays](https://www.ericsdevblog.com/index.php/2021/09/javascript-basics-3-arrays-and-objects/#Arrays)
  + [Array Loops](https://www.ericsdevblog.com/index.php/2021/09/javascript-basics-3-arrays-and-objects/#Array_Loops)
  + [Stacks and Queues](https://www.ericsdevblog.com/index.php/2021/09/javascript-basics-3-arrays-and-objects/#Stacks_and_Queues)
* [Properties and Methods](https://www.ericsdevblog.com/index.php/2021/09/javascript-basics-3-arrays-and-objects/#Properties_and_Methods)
* [Objects](https://www.ericsdevblog.com/index.php/2021/09/javascript-basics-3-arrays-and-objects/#Objects)
* [JSON](https://www.ericsdevblog.com/index.php/2021/09/javascript-basics-3-arrays-and-objects/#JSON)

**ARRAYS**

Let’s first talk about arrays. An array is a list of values wrapped inside a pair of square brackets, separated by commas.

let listOfNumbers = [1, 2, 3, 4, 5];

let listOfStrings = ["1", "2", "3", "4", "5"];

We can access the elements in the array by referring to its index number, which starts from 0 instead of 1. This is a very common thing in the computer world, you should get used to it.

let x = listOfNumbers[0]; // x = 1 (index 0 is the first element)

let y = listOfNumbers[2]; // y = 3 (index 3 is the third element)

What if we want to access the last element of the array, and we don’t know the length of the array? Instead of trying every index starting from 0 to see if it gives a value, we can access the length of the array using arrayName.length. The length here is called property and . is how we can access the property of a value. For example:

let z = listOfNumbers[listOfNumbers.length - 1]; // z = 5

In this example, listOfNumbers.length gives us 5, because we start from 1 when we calculate length. However, since the index always starts from 0, that means the index of the last element should be 1 less than the length, hence the -1 here.

It is also possible to change the values in the array.

let listOfNumbers = [1, 2, 3, 4, 5];

listOfNumbers[2] = 100;

// listOfNumbers = [1, 2, 100, 4, 5];

**Array Loops**

In some cases, we may need to iterate over the entire array and access each element one by one. There are two different ways we can do this.

let list = [...];

for (let e of list) {

...

// Do somthing with the variable e

}

This is the method we’ve seen before, for every iteration, the variable e will be assigned the next element in the array list, and you can do something with the variable e inside that for loop.

The second method is to use the index.

for (let i = 0; i < list.length; i++) {

let e = list[i];

}

In this example, the variable i is bound to the index of the array elements, and e is bound to the array element with the index i. The expression i++ is a shorthand notation of i = i + 1.

**Stacks and Queues**

Stacks and queues are two very important data structures in computer programming, which we can implement using arrays in JavaScript.

A stack is a structure of elements based on the principle of last in first out (LIFO). It is like a stack of books. If you want to add a new book to the stack, you would put it on the top, and if you want to remove a book, you would remove the one on the top as well.

A picture containing text, furniture

Description automatically generated

The stack data structure has two fundamental operations:

1. The **push** operation is responsible for inserting (pushing) a new element to the stack.
2. The **pop** operation is responsible for removing and returning the most recent element from the stack.

Luckily, JavaScript offers these two methods out of the package. To use them we can, again, use the . operator, because methods are properties with function values:

let stack = [];

stack.push(2);

stack.push(5);

stack.push(7);

stack.push(9);

stack.push(7);

console.log(stack); // -> [2, 5, 7, 9, 7]

stack.pop(); // -> returns 7

stack.pop(); // -> returns 9

stack.pop(); // -> returns 7

stack.pop(); // -> returns 5

stack.pop(); // -> returns 2

console.log(stack); // -> []

Graphical user interface, text, application

Description automatically generated

A queue is another very useful data structure. It is very similar to stacks, except it follows the first in first out (FIFO) principle. It’s like waiting in line in a restaurant, if you come first, you will get the food first.

Table

Description automatically generated

The queue data structure has two fundamental operations as well:

1. The **enqueue** operation is responsible for inserting a new element at the end of the queue.
2. The **dequeue** operation is responsible for removing and returning the element at the beginning of the queue.

There are also two methods built into JavaScript that help us with these two operations, however, the terminologies are a bit different. For the enqueue operation, we can use the push() method, because it pushes the new element to the end of the array. As for the dequeue operation, we can use the shift() method, which removes the first element of the array.

let queue = [];

queue.push(2);

queue.push(5);

queue.push(7);

queue.push(9);

queue.push(7);

console.log(queue);

queue.shift();

queue.shift();

queue.shift();

queue.shift();

queue.shift();

Graphical user interface, application

Description automatically generated

**PROPERTIES AND METHODS**

Now, let’s take a closer look at the concept of property that we were just talking about. We’ve seen some strange-looking expressions like listOfNumbers.length and Math.max. These are expressions that access the property of some value. In the first example, we access the length property of the listOfNumbers array. In the second example, we access the max property in the Math object.

Almost all of the data types we talked about have built-in properties. For example, a string also has a length property just like arrays, which store the length of that string.

In addition to the length property that only holds a number, there are also a number of properties that hold function values. For instance, we could use the toUpperCase property of a string to get a copy of that string in which all letters in that string are converted to uppercase. We usually refer to these properties with function values as methods.

let aString = "abCDefg";

console.log(aString.toUpperCase()); // -> "ABCDEFG"

console.log(aString); // -> "abCDefg"

**OBJECTS**

All of the properties we’ve talked about so far are built-in properties, and they all come with JavaScript. But, what if we want to create our own properties? Objects are the second datatype we are going to discuss here which allows us to create our own properties.

An object is an arbitrary collection of properties, defined using the curly braces {}. For example, here we define an object named MyHousehold:

let MyHousehold = {

members: 4,

names: ["Mason", "Robert", "Lara", "Wynne"]

};

Inside the braces is a list of properties separated by commas. Each property is defined in name: value format.

In this example, there are four members in my household. To access that information, we can use the same notation we used before, with a dot (.).

console.log(MyHousehold.members);

// -> 4

The objects are also mutable, which means their values can be modified. We can do that by using the = operator. For example:

MyHousehold.members = 5;

console.log(MyHousehold.members);

// -> 5

However, this only applies to the object data type. The properties of the other data types, such as strings, numbers, and arrays, their properties are immutable, and cannot be modified. For instance, if you have a string that contains "cat", it is not possible for you to write some code to change a character in that string to make it spell "rat".

In the next article, we’ll dig deeper into objects, and talk about the concept of object-oriented programming.

**JSON**

When you define a property (name: value), the name does not actually contain its value. Instead, it is stored in the computer memory as an address, which points to the location in the memory where the value is stored.

If you want to save the data in a file or send it to someone else over the internet, you’ll have to somehow convert these tangles of memory address into a description that can be stored or sent over the internet. This process is called serialization, this means the data is converted into a flat description. A popular serialization format is called JSON(JavaScript Object Notation, pronounced “Jason”).

JSON looks just JavaScript’s way of defining objects, with a few extra restrictions. The property names have to be surrounded by double quotes, and no functions or anything that involves actual computation, only the simple data types are allowed. So, if we express our MyHousehold object in JSON format, it would look like this:

{

"members": 4,

"names": ["Mason", "Robert", "Lara", "Wynne"]

}

JSON is widely used as a data storage and communication format on the web, even in languages other than JavaScript. We will encounter it again as we talk about backend development in the future.