**Arrays**

**Intro to Arrays**

Array is a data structure that allows us to store multiple values of different data types into a variable.

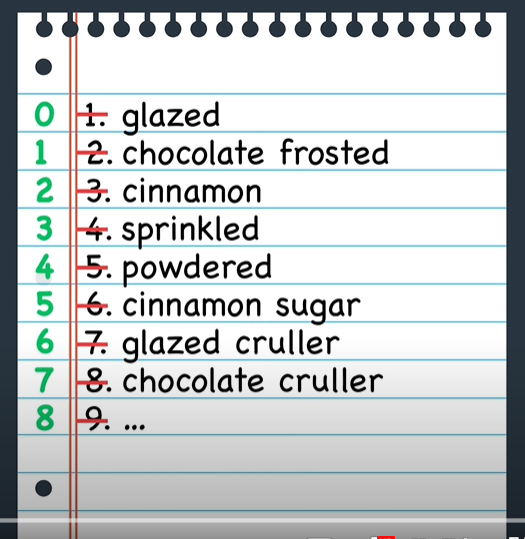
Let’s see how to create, use arrays and how they are structured.

**Donuts to Code**

To store donut names, we can either create multiple variables - which is a mess, hard to manage all variables - or, we can use a single variable of array.

**Array**: a data structure used to store multiple values. The values stored in the array are ordered.

Array is like a numbered list (starting from 0), where each item in the list has a number, and we can use that number to refer back to the item when we need it:



**Creating an Array**

An array is useful because it stores multiple values into a single, organized data structure.

To define an array we list all values separated with commas between square brackets:

// creates a `donuts` array with three strings

var donuts = ["glazed", "powdered", "jelly"];

We can store any data type in array, not only strings:

// creates a `mixedData` array with mixed data types

var mixedData = ["abcd", 1, true, undefined, null, "all the things"];

We can even store an array in an array to create a **nested array**:

// creates a `arraysInArrays` array with three arrays

var arraysInArrays = [[1, 2, 3], ["Julia", "James"], [true, false, true, false]];

**Accessing Array Elements**

An element in arrays means each individual piece of data in an array.

* **Index**: position of an element in an array.

Just like strings, each element of an array is numbered starting from 0, and we can those numbers (indexes) to access an element in the array:



The syntax used to access a particular element in the array is: name + square brackets + index inside brackets.

var donuts = ["glazed", "powdered", "jelly"];

console.log(donuts[0]);

**Array Index**

Elements in an array are indexed starting at the position 0.

If we try to access an element at an index that does not exist, a value of undefined will be returned back.

Indexes can also be useful to edit the elements. In case we want to modify a particular element inside an array we simply assign a new value by passing the index to be edited:

donuts[1] = "glazed cruller";

**Array Properties and Methods**

Arrays have a number of properties and methods that make them powerful data structures.

* Properties: special pieces of information about data structure. For example, the property *length* used to get the number of elements in an array.
* Methods: special predefined functions that a data structure can call. Common array methods are: reverse (reverses the order of elements in an array), sorts (sorts the elements in an array), push and pop (add and remove elements in an array).

For a full list of all built-in array methods, check out the link: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

**Length Property**

By using the length property we can find how many elements an array contains.

var donuts = ["glazed", "powdered", "sprinkled"];

console.log(donuts.length);

**Push Method**

To add a new element to the array, we can use the push method. push() adds a new element to the end of the array:

var donuts = ["glazed", "chocolate frosted", "Boston creme", "glazed cruller", "cinnamon sugar", "sprinkled"];

donuts.push("powdered"); // pushes "powdered" onto the end of the `donuts` array

console.log(donuts)

Inside the push parentheses we add the value we want to add to the array.

**Pop**

To remove an element from the end of an array, we can use the pop method. pop() removes the last element from the array:

var donuts = ["glazed", "chocolate frosted", "Boston creme", "glazed cruller", "cinnamon sugar", "sprinkled", "powdered"];

donuts.pop(); // pops "powdered" off the end of the `donuts` array

donuts.pop(); // pops "sprinkled" off the end of the `donuts` array

donuts.pop(); // pops "cinnamon sugar" off the end of the `donuts` array

console.log(donuts)

We don’t need to pass the value inside the pop parentheses. However, the pop method will always remove the last element of the array.

**Splice Method**

Splice() is another handy method that allows us to add or remove elements from anywhere within an array.

While push() and pop() limit us to adding and removing elements from the end of an array, splice() let’s us specify the index location to add and remove elements, as well the number of elements:

var donuts = ["glazed", "chocolate frosted", "Boston creme", "glazed cruller"];

donuts.splice(1, 1, "chocolate cruller", "creme de leche");

// removes "chocolate frosted" at index 1 and adds "chocolate cruller" and "creme de leche" starting at index 1

**Splice() Syntax**

**arrayName.splice(arg1, arg2, item1, ....., itemX);**

**Arg1**: mandatory argument. Specify the index position to add or delete elements. When using a negative value, it counts from the end of the array. Eg. -1 specifies the last element.

**Arg2**: optional argument. Specify the number of elements to be removed. If set to 0, no items will be removed.

**Item1, ..., itemx**: are the items to be added at the index position **arg1**.

[MDN documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice), here we can find more examples of splice().

**Array Loops**

Once the data is in the array, we are able to access and manipulate each element in the array without writing repetitive code for each element.

For instance, if we want to add “ hole” to the end of each element, we either need to add one by one:

var donuts = ["jelly donut", "chocolate donut", "glazed donut"];

donuts[0] += " hole";

donuts[1] += " hole";

donuts[2] += " hole";

Or, we can use for loops:

var donuts = ["jelly donut", "chocolate donut", "glazed donut"];

// the variable `i` is used to step through each element in the array

for (var i = 0; i < donuts.length; i++) {

donuts[i] += " hole";

donuts[i] = donuts[i].toUpperCase();

}

***donuts array:*** *["JELLY DONUT HOLE", "CHOCOLATE DONUT HOLE", "GLAZED DONUT HOLE"]*

**The forEach() loop**

Arrays have a set of special methods to help us iterate over and perform operations on collections of data:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

A method that can help us with iteration is:

* **forEach()** : an alternative way to iterate over an array, and manipulate each element in the array with an inline function expression.

var donuts = ["jelly donut", "chocolate donut", "glazed donut"];

//using conventional for loop:

for (var i = 0; i < donuts.length; i++) {

donuts[i] += " hole";

donuts[i] = donuts[i].toUpperCase();

console.log(donuts[i]);

}

//using forEach() method:

donuts.forEach(function(donut) {

donut += " hole";

donut = donut.toUpperCase();

console.log(donut);

});

The forEach() method iterates over the array without the need of an explicitly defined index.

The forEach() method must be used when looping over every element from start to finish.

**The Map() Method**

With the [map() method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map), we can take an array, perform some operation on each element of the array, and return a new array.

var donuts = ["jelly donut", "chocolate donut", "glazed donut"];

var improvedDonuts = donuts.map(function(donut) {

donut += " hole";

donut = donut.toUpperCase();

return donut;

});

console.log(donuts, improvedDonuts);

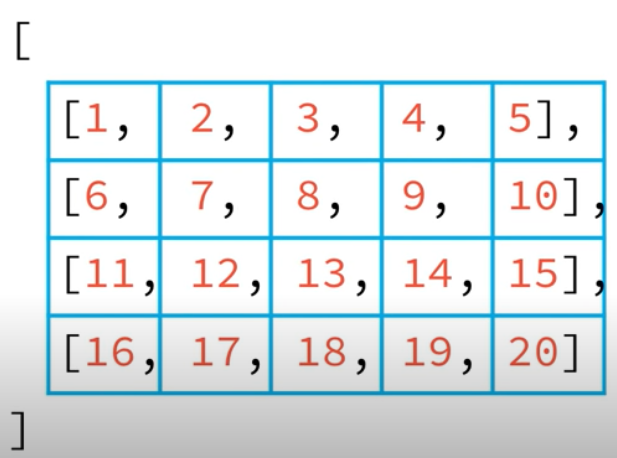
The map() method accepts one argument, which is a function used to manipulate each element in the array. It’s different from the forEach method, as we no longer need to iterate over the indices anymore.

**Arrays in Array**

We can store pretty much any data in an array, even arrays inside an array.

Array with arrays is a two dimensional array, and iterating over it can be tricky.

It’s helpful to think about the whole array as a grid, where each row of the array is an array and each of those arrays are elements of the larger array:



A single loop can access each ‘row of the grid’, while a nested loop can access each element of each row.

**2D Donut Arrays**

We could use an array of arrays that has the name of each donut associated with its position in the box. Here is an example:

var donutBox = [

["glazed", "chocolate glazed", "cinnamon"],

["powdered", "sprinkled", "glazed cruller"],

["chocolate cruller", "Boston creme", "creme de leche"]

];

console.log(donutBox, donutBox[1][2]);

In order to access each element of each array, we need use a inner loop:

donutBox.forEach(function(row\_donutBox, i) {

// console.log(i, row\_donutBox);

row\_donutBox.forEach(function(column\_donutBox, column\_i){

console.log(i, column\_i, column\_donutBox);

})

});

f