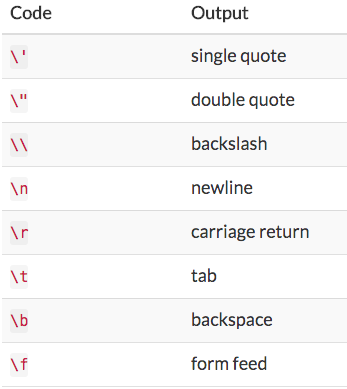
FCC- Javascript

In computer science, *data* is anything that is meaningful to the computer. JavaScript provides seven different *data types* which are undefined, null, boolean, string, symbol, number, and object.

* 

Strings:

* : So if var firstName = "Charles," you can get the value of the first letter of the string by using firstName[0]
* In JavaScript, String values are *immutable*, which means that they cannot be altered once created.
  + Note that this does *not* mean that myStr cannot be changed, just that the individual characters of a *string literal* cannot be changed. The only way to change myStr would be to assign it with a new string, like this

var myStr = "Bob";  
myStr = "Job";

* In order to get the last letter of a string, you can subtract one from the string's length.
* For example, if var firstName = "Charles", you can get the value of the last letter of the string by using firstName[firstName.length - 1]

Arrays

* You can also nest arrays within other arrays, like this: [["Bulls", 23], ["White Sox", 45]]. This is also called a Multi-dimensional Array.
* We can access data inside an array using indexes. Array indexes are written in the same bracket notation that strings use, except that instead of specifying a character, they are specifying an entry in the array.
* Create a variable called myData and set it to equal the first value of myArray using bracket notation.

var myData = myArray[0]

* One way to think of a multi-dimensional array, is as an array of arrays.

**Example**

var arr = [  
  [1,2,3],  
  [4,5,6],  
  [7,8,9],  
  [[10,11,12], 13, 14]  
];  
arr[3]; // equals [[10,11,12], 13, 14]  
arr[3][0]; // equals [10,11,12]  
arr[3][0][1]; // equals 11

* An easy way to append data to the end of an array is via the push()function.
* .push() takes one or more parameters and "pushes" them onto the end of the array.

var arr = [1,2,3];  
arr.push(4);  
// arr is now [1,2,3,4]

* Another way to change the data in an array is with the .pop()function.
* .pop()is used to "pop" a value off of the end of an array. We can store this "popped off" value by assigning it to a variable. In other words, .pop()removes the last element from an array and returns that element.

var threeArr = [1, 4, 6];  
var oneDown = threeArr.pop();  
console.log(oneDown); // Returns 6  
console.log(threeArr); // Returns [1, 4

* pop() always removes the last element of an array. What if you want to remove the first?
* That's where .shift()comes in. It works just like .pop(), except it removes the first element instead of the last.
* Not only can you shift elements off of the beginning of an array, you can also unshift elements to the beginning of an array i.e. add elements in front of the array.
* .unshift()works exactly like .push(), but instead of adding the element at the end of the array, unshift()adds the element at the beginning of the array.

Function

* Variables which are used without the var keyword are automatically created in the global scope. This can create unintended consequences elsewhere in your code or when running a function again. You should always declare your variables with var.
* It is possible to have both local and global variables with the same name. When you do this, the localvariable takes precedence over the globalvariable.
* In JavaScript, you can determine the type of a variable or a value with the typeof operator, as follows:
* typeof 3 // returns 'number'  
  typeof '3' // returns 'string'

Switch Statements and if/else

* Since === returns true or false, we can return the result of the comparison:

function isEqual(a,b) {

return a === b;

}

## Basic JavaScript: Build JavaScript Objects

Objects are similar to arrays, except that instead of using indexes to access and modify their data, you access the data in objects through what are called properties. Objects are useful for storing data in a structured way, and can represent real world objects, like a cat.

Heres a sample cat object:

var cat = {  
  "name": "Whiskers",  
  "legs": 4,  
  "tails": 1,  
  "enemies": ["Water", "Dogs"]  
};

There are two ways to access the properties of an object: dot notation (.) and bracket notation ([]), similar to an array.

Objects can be thought of as a key/value storage, like a dictionary. If you have tabular data, you can use an object to "lookup" values rather than a switch statement or an if/else chain. This is most useful when you know that your input data is limited to a certain range.

## Basic JavaScript: Manipulating Complex Objects

Sometimes you may want to store data in a flexible Data Structure. A JavaScript object is one way to handle flexible data. They allow for arbitrary combinations of strings, numbers, booleans, arrays, functions, and objects.

Here's an example of a complex data structure:

var ourMusic = [  
  {  
    "artist": "Daft Punk",  
    "title": "Homework",  
    "release\_year": 1997,  
    "formats": [   
      "CD",   
      "Cassette",   
      "LP"  
    ],  
    "gold": true  
  }  
];

This is an array which contains one object inside. The object has various pieces of metadata about an album. It also has a nested "formats"array. If you want to add more album records, you can do this by adding records to the top level array.

Objects hold data in a property, which has a key-value format. In the example above, "artist": "Daft Punk"is a property that has a key of "artist" and a value of "Daft Punk".

[JavaScript Object Notation](http://www.json.org/) or JSONis a related data interchange format used to store data.

## Basic JavaScript: Iterate with JavaScript While Loops

The first type of loop we will learn is called a "while" loop because it runs "while" a specified condition is true and stops once that condition is no longer true.

var ourArray = [];  
var i = 0;  
while(i < 5) {  
  ourArray.push(i);  
  i++;  
}

FOR LOOPS

You can run the same code multiple times by using a loop. The most common type of JavaScript loop is called a "for loop" because it runs "for" a specific number of times. For loops are declared with three optional expressions separated by semicolons:

for ([initialization]; [condition]; [final-expression])

## Iterate Through an Array with a For Loop

A common task in JavaScript is to iterate through the contents of an array. One way to do that is with a for loop. This code will output each element of the array arr to the console:

var arr = [10,9,8,7,6];  
for (var i = 0; i < arr.length; i++) {  
   console.log(arr[i]);  
}

Remember that Arrays have zero-based numbering, which means the last index of the array is length - 1. Our condition for this loop is i < arr.length, which stops when I is at length - 1.

## Basic JavaScript: Nesting For Loops

If you have a multi-dimensional array, you can use the same logic as the prior waypoint to loop through both the array and any sub-arrays. Here is an example:

var arr = [  
  [1,2], [3,4], [5,6]  
];  
for (var i=0; i < arr.length; i++) {  
  for (var j=0; j < arr[i].length; j++) {  
    console.log(arr[i][j]);  
  }  
}

This outputs each sub-element in arr one at a time. Note that for the inner loop, we are checking the .length of arr[i], since arr[i] is itself an array.

## Iterate with JavaScript Do...While Loops

You can run the same code multiple times by using a loop.

The next type of loop you will learn is called a "do...while" loop because it first will "do" one pass of the code inside the loop no matter what, and then it runs "while" a specified condition is true and stops once that condition is no longer true. Let's look at an example.

var ourArray = [];  
var i = 0;  
do {  
  ourArray.push(i);  
  i++;  
} while (i < 5);

This behaves just as you would expect with any other type of loop, and the resulting array will look like [0, 1, 2, 3, 4]. However, what makes the do...while different from other loops is how it behaves when the condition fails on the first check. Let's see this in action.

Essentially, a do...while loop ensures that the code inside the loop will run at least once.

Let's try getting a do...while loop to work by pushing values to an array.

Change the while loop in the code to a do...while loop so that the loop will push the number 10 to myArray, and I will be equal to 11 when your code finishes running.

## Generate Random Fractions with JavaScript

Random numbers are useful for creating random behavior.

JavaScript has a Math.random() function that generates a random decimal number between 0(inclusive) and not quite up to 1(exclusive). Thus Math.random()can return a 0 but never quite return a 1

Use another function, Math.floor()to round the number down to its nearest whole number.

Math.floor(Math.random() \* 20); //generates a whole number from 0-19

Instead of generating a random number between zero and a given number like we did before, we can generate a random number that falls within a range of two specific numbers.

To do this, we'll define a minimum number min and a maximum number max.

## Basic JavaScript: Use the parseInt Function

The parseInt() function parses a string and returns an integer. Here's an example:

var a = parseInt("007");

WITH A RADIX

If the first character in the string can't be converted into a number, then it returns NaN

The parseInt() function parses a string and returns an integer. It takes a second argument for the radix, which specifies the base of the number in the string. The radix can be an integer between 2 and 36.

The function call looks like:

parseInt(string, radix);

And here's an example:

var a = parseInt("11", 2);

The radix variable says that "11" is in the binary system, or base 2. This example converts the string "11" to an integer 3.

## Use the Conditional (Ternary) Operator

The conditional operator, also called the ternary operator, can be used as a one line if-else expression.

The syntax is:

condition ? statement-if-true : statement-if-false;

MULTIPLE

function findGreaterOrEqual(a, b) {  
  if(a === b) {  
    return "a and b are equal";  
  }  
  else if(a > b) {  
    return "a is greater";  
  }  
  else {  
    return "b is greater";  
  }  
}

The above function can be re-written using multiple conditional operators:

function findGreaterOrEqual(a, b) {  
  return (a === b) ? "a and b are equal" : (a > b) ? "a is greater" : "b is greater";  
}