Basics

&& : and
|| : or
! : not
1 : true
0 : false
increments:
i++ , i-- , i += x , i -= x
isNaN('berry'); -- true -- returns true when not a number
isNaN(42); -- false
x**.length** -- Returns the length of a string or the amount of items in a variable **typeof** variable -- returns object, number, string
Z < a -- true, uppercase < lowercase

```
// In browsers, the global scope object is stored in the window variable:
var myVar = 10; // global variable
console.log("myVar" in window); // true
console.log(window.myVar); // 10
```

Arrays & Objects:

```
var newArray = []; // could do variable = [1,2,3] as well
newArray.push('hello');
newArray[0]; // equals  'hello;
var newObject = {
  key: value,
  key: value
} // object literal notation
// Each piece of information we include in an object is known as a property. Each Property has a Value
var myObj = new Object(); // object constructor
myObj["key"] = "value"; // or myObj.key = "value";

delete newObject.key; // removes the key (very uncommon)
console.log("key" in anObject); // "in" returns a Boolean value that indicates whether that object has that property

myObj.hasOwnProperty('name') // returns true if myObj has a name property
```

Functions:

```
var newFunction = function(argument,argument) { }; // or function newFunction(){} = function declarations = not part of the regular
newFunction(x,y);
// below are 3 new things for functions in JavaScript 6 (experimental for now)
argument => argument + 1; // is equal to function(argument){ return argument + 1; };
function(x = 7, y = 42) { } // to set default values
function(x, y, ...a) { } // ...a will store all the
var newFunction = function() { return "hi" };
newFunction // returns the function while newFunction() would return the functions outcome.
```

Recursive Functions

```
// Functions calling themselves (create a kindof finite loop)
function power(base, exponent) {
  if (exponent == 0)
    return 1;
  else
    return base * power(base, exponent - 1); // don't forget to return your recursion else, you'll get an undefined.
}

console.log(power(2, 3));
// → 8
```

Simplest example:

```
(function loop(base, i){
  // do something
  if(i>0) return loop(base, i-1);  // loop
})(["a", "b"], length)
```

```
/*
 * Whenever a function is called, a special variable named arguments is added to the environment in which the function body runs.
 */
function argumentCounter() {
  console.log("You gave me", arguments.length, "arguments.");
}
argumentCounter("Straw man", "Tautology", "Ad hominem"); // You gave me 3 arguments.
```

Methods:

```
 var bob = new Object();
bob.age = 17; // or: bob["age"] = 17; this way you can use variables as well: bob[variable];
bob.setAge = function (newAge){ bob.age = newAge; };
bob.setAge(5); // would set the Age Property of bob to 5 using the respective function
// here we define our method using "this", before we even introduce bob
var setAge = function (newAge) { this.age = newAge; };
var bob = { age: 25, setAge: setAge }
bob.setAge(50);
// writing an object constructor or object literal notation – both work the same
function someObject() {
  this.stuff = "bla"
  this.someMethod = function() { console.log(this.stuff) };
}

// Associate methods to properties
var pile = {
  elements: ["eggshell", "orange peel", "worm"],
  get height() {
    return this.elements.length;
  },
  set height(value) {
    console.log("Ignoring attempt to set height to", value);
  }
};
// the get and set notation for properties allows you to specify a function to be run when the property is read or written
console.log(pile.height); // → 3
pile.height = 100; // → Ignoring attempt to set height to 100

// You can also add such a property to an existing object, for example a prototype, using the Object.defineProperty function
function Pile() {
  this.elements = ["eggshell", "orange peel", "worm"];
}

Object.defineProperty(Pile.prototype, "height", {
  get: function() { return this.elements.length; }
});

var test = new Pile();
console.log(test.height); //-> 3
```

Predefined Functions/Methods

On Strings

```
/*
 * toUpperCase, toLowerCase, trim, charAt
 */
"Doh".toUpperCase() // "DOH"
"Doh".toLowerCase() // "doh"
"  okay \n ".trim() // "okay"
"bla".charAt(0) // b -- identical to "bla"[0]

/*
 * Repeat
 */
'-'.repeat(2); //-> '--' (not supported by IE and Opera)
```

On Arrays

```
/*
 * Push, Pop, shift and unshift
 */
var mack = [];
mack.push("Mack");
mack.push("the", "Knife");
mack; // → ["Mack", "the", "Knife"]
mack.join(" "); // → Mack the Knife
mack.pop(); // → Knife
mack; // → ["Mack", "the"]

// push and pop add or remove elements at the end of an array
// shift and unshift add or remove elements at the start of an array

/*
 * indexOf, lastIndexOf
 */
[1, 2, 3, 2, 1].indexOf(2); // → 1
[1, 2, 3, 2, 1].lastIndexOf(2); // → 3

/*
 * slice & concat
 */
[0, 1, 2, 3, 4].slice(2, 4); // → [2, 3] (positions 2,4 – 2 included 4 excluded)
[0, 1, 2, 3, 4].slice(2); // → [2, 3, 4]
["a", "b"].concat("c", 1, "d") // → ["a", "b", "c", 1, "d"]

/*
 * filter
 */
function isBigEnough(value) { return value >= 10; }
var filtered = [12, 5, 8, 130, 44].filter(isBigEnough); // → filtered is [12, 130, 44]

/*
 * map
 */
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt); // roots is now [1, 2, 3] (1*1=1, 2*2=4, 3*3=9), numbers is still [1, 4, 9]

/*
 * reduce
 */
var total = [0, 1, 2, 3].reduce(function(a, b) { return a + b; }); // total == 6

/*
 * forEach
 */
function logArrayElements(element, index, array) { console.log('a[' + index + '] = ' + element); }
[2, 5, , 9].forEach(logArrayElements); // → a[0] = 2 // → a[1] = 5 // → a[3] = 9
// Notice that index 2 is skipped since there is no item at that position in the array.

/*
 * every & some
 */
function isBigEnough(element, index, array) { return element >= 10; }
[12, 5, 8, 130, 44].every(isBigEnough);   // false
[12, 5, 8, 130, 44].some(isBigEnough);   // true
```

Nodes

```
/*
 * Select children of an element with content
 * given: table > tr > td + td
 */
var t = document.querySelector('#table');
var td = t.content.querySelectorAll("td"); // nodelist [td, td]
// note: without .content it would not be possible to select the childs.

/*
 * clone nodes with importNode
 * see https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template for a good example
 */
var clone = document.importNode(externalNode, deep);

var tb = document.getElementsByTagName("tbody");
var clone = document.importNode(t.content, true); // import/clone the content of the table
tb[0].appendChild(clone); // append it to tbody
```

Custom Constructors / Classes:

```
// new Object(); is a predefined constructor by js that creates an empty object, we can create our own class constructors like so:
function Person(name,age) {
  this.name = name;
  this.age = age;
}
var bob = new Person("Bob Smith", 30); // creates an object/class Person with the keys/properties specified in our constructor
// here is where a constructor notation makes sence

function Dog (breed) {
  this.breed = breed;
};

var buddy = new Dog("golden Retriever");
Dog.prototype.bark = function() { console.log("Woof"); }; // prototype adds that method to the constructor. Instead of buddy.bark =
buddy.bark();
var snoopy = new Dog("Beagle");
snoopy.bark(); // is also able to bark.

function Penguin(breed) {
    this.breed = breed;
    var priv = "Private Variable"; // a variable that is only accessible within the constructor
    this.getPriv = function() { return priv; }; // to be able to access a private variable from outside
}
Penguin.prototype = new Dog(); // Penguins will inherit from Dog and also be able to bark. A Prototype Chain.
var penguin = new Penguin("penguin"); // create an instance of the Penguin class.
penguin.bark(); // Woof

/*
 * defineProperty & hasOwnProperty
 */
Object.prototype.nonsense = "hi"; // hi will be an enumerable property, hence show up in any for .. in .. loop on any object. To pre
for (var name in map) console.log(name); // → pizza // → touched tree // → nonsense
delete Object.prototype.nonsense;
Object.defineProperty(Object.prototype, "hiddenNonsense", {enumerable: false, value: "hi"});
for (var name in map) console.log(name); // → pizza // → touched tree
console.log(map.hiddenNonsense); // → hi
// alternatively:
for (var name in map) {
  if (map.hasOwnProperty(name)) {
    // ... this is an own property
  }
}
```

Math:

```
// returns the largest of 2 numbers
Math.max(10, 20) //-> 20

// returns the largest integer less than or equal to a given number
Math.floor( 45.95); //-> 45

/*
 * Randoms
 */
// random Number between 0 and 1 but never 0 nor 1
Math.random() //-> i.e. 0.73452221
// *x sets the width and +y the range of the randomness
Math.floor(Math.random() * 5 + 1) // Random Number between 1-5
// Returns a random integer between min (included) and max (included)
// Using Math.round() will give you a non-uniform distribution!
function getRandomIntInclusive(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

/*
 * .sqrt to calculate distances
 */
// Pythagorean theorem says that the square of the distance we are looking for is equal to the square of the x-coordinate plus the s
Math.sqrt(a * a + b * b);
// or less chic: Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
```

Loops:

```
for(var i = 0; i < 6; i++){ }; // for a number of time
for(var p in obj){ } // for in iterates a specified variable p over all the enumerable properties of an object
while(true){ }; // while a condition is true
do{ }while(true); // does x while a condition is true (runs x at least once)
break; // terminate a loop, switch, or in conjunction with a label statement.
["a", "b", "c"].forEach(function(entry) { console.log(entry); }); //although move convenient, forEach is ~30% slower than a normal f
```

Logical Operators

```
if(true){ /* */ } else if (true){ /* */ } else { /* */ }
```

```
switch(variable){ case 'option1': /* */ break; case 'option2': /* */ break; default: /* */ }
```

Conditional Operator

```
console.log(true ? 1 : 2); // → 1
console.log(false ? 1 : 2); // → 2
// The value before ? “picks” which of the other two values will come out. When true, the first value is chosen, and when false, the
```

User Input

```
var name = prompt("who're you?");
```

JSON

```
// JSON.stringify takes a JavaScript value and returns a JSON-encoded string
var string = JSON.stringify({name: "X", born: 1980});
console.log(string); // → {"name":"X","born":1980}
// JSON.parse takes such a string and converts it to the value it encodes
console.log(JSON.parse(string).born); // → 1980
```

ES6

```
/*
 * Using glob variables in new objects
 */
var foo = 2;
var obj = { bar: 1, foo } //-> obj.bar = 1 -> obj.foo = 2
```

Destructuring

```
var foo = {
  bar: 1,
  baz: 2
};
var { bar, baz } = foo;
console.log(bar); //-> 1
console.log(baz); //-> 2 // instead of foo.bar / foo.baz
// also with arrays:
var array = [1, 2];
var [ a, b ] = array;
console.log(a,b) //-> 1 2
// very nice on functions:
// usually you would have an opts value but now
// you can destructure the incoming object directly in the function
// also note how we changed the name of weight to just w
function calcBmi({ weight: w, height, max, callback }){
  var bmi = w / Math.pow(height, 2);
  if(bmi > max) { console.log('sorry you are obese') }
  if(callback) { callback(bmi) } // instead of if objs.callback
}
// by passing objects
calcBmi({ weight, height, max: 25 });
// you can now omit a data and it will just be passed as 'undefined', here we leave out max.
// also the order does not matter anymore!
calcBmi({ callback: function() {}, weight, height });
```

Default Arguments

```
function someThing(value = 12){
  // stuff
}
```

Template Strings

```
var name = "will";
var thing = "party";
// old way
var greet = "hi, my name is" + name + "\n and I like to" + thing + "!";
// new way. Note the inline variables and how to break lines.
var greet = `hi, my name is ${name}
             and I like to ${thing} !`;
```

Scoping

```
// let is the new var
// define block variables (only used within blocks)
if () { let b = 2; }
console.log(b); // undefined

// use const as default and change it to let only when you have to change the value
// const is also block scoped but unchangable
const a = 1;
a = 2; //-> error
```

Classes

```
// old:
function Parent() { /*constructor*/ }
Parent.prototype.foo = function() {}; // to add a function to the constructor
var parent = new Parent();
parent.foo();
// new
class Parent {
  // in ES7 I could also add static properties here, like this:
  age = 30;

  constructor() {

  }

  foo() {

  }

  static bar() {

  }
}
var parent = new Parent();
parent.foo();
parent.age; //-> 30
Parent.bar();

class Child extents Parent {
  constructor() {
    super() // will call the constructor of Parent
  }
  baz() {

  }
}
var child = new Child();
child.baz(); // works
child.foo(); // works
```

Arrow Functions

```
// basics
// old
var foo = function(a, b) { return a + b; }
// new
var foo = (a, b) => { return a + b; }

// implicit returns (only on one liners):
do.something((a, b) => a + b);

// also without parenthesis when you only have one argument
do.something(a => a + 1);
[0,1,2].map(val => val+1); //-> [1,2,3]

// it also binds the this
var module = {
  age: 30,
  foo: function() {
    setTimeout(() => { // automatically bind the previous this to the function
      console.log(this.age); // still works
    }, 100);
  }
};
// note that it will overwrite i.e. jQuerys this value.
```

Modules

```
// old
module.exports.foo = function () {};
module.exports.bar = function () {};
// another file:
var myModule = require("myModule");
var foo = myModule.foo;
// new
export function foo() { };
export function bar() { };
export var x = 3;
import { foo, bar, x as test } from "myModule";
console.log(test) //-> 3
// or whole modules
export default {};
import myModule from "myModule";
```

Generator Functions

```
async function() {
  var friends = await $.get("http://bla.com/friends")
  console.log(friends);
} // returns one promise that is denible
```

Tricks

Get password combinations

```
var passwords = function(chars, length){
    var chars = chars.split("");
    var index=[];

    (function loop(base, i){
        for(var k=0; k< chars.length; k++) {
            if(i>1) loop(base+chars[k], i-1);
            else index.push(base+chars[k]);
        }
    })("", length)

    return index;
}

var possible = "0123456789";
var length = 4
var possibilities = passwords(possible, length);
```

Use them

```javascript var length = possibilities.length; (function loop(base, i){ if(i < 0) return; let n = base[i]; i -= 1; setTimeout(() => { return loop(base, i) }); })(possibilities, length);
```