JAVASCRIPT

High-level, weakly typed, dynamic, prototype-based, multi-paradigm and interpreted programming language
Alongside HTML and CSS, JAVASCRIPT is one of the three core technologies of the World Wide Web
Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of software, including server-side in web servers and databases, non-web programs and runtime environments
Node.js is an example of an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser
Although there are strong outward similarities between JavaScript and Java, the two languages are distinct and differ greatly in design

JavaScript / ECMAScript
Ecma International is an organization that creates standards for technologies
ECMA-262 is a standard published by Ecma International, containing the specification for a general purpose scripting language, specifically for acting on an existing entity or system
ECMAScript is the specification defined in ECMA-262 for creating a general purpose scripting language
ECMAScript provides the rules, details, and guidelines that a scripting language must observe to be considered ECMAScript compliant
JavaScript is a general purpose scripting language that conforms to the ECMAScript specification
ECMAScript specification is how to create scripting language
JavaScript documentation is how to use scripting language
JavaScript engines are commonly found in web browsers, including V8 in Chrome, SpiderMonkey in Firefox, and Chakra in Edge and each engine is like a language module for its application, allowing it to support a certain subset of the JavaScript language
Releasing a new edition of ECMAScript does not mean that all JavaScript engines in existence suddenly have those new features
It is up to the groups or organizations who are responsible for JavaScript engines to be up-to-date about the latest ECMAScript specification, and to adopt its changes
If a new edition of ECMAScript comes out, JavaScript engines do not integrate the entire update at one go, they incorporate the new ECMAScript features incrementally
JavaScript was invented by Brendan Eich in 1995 and submitted to Ecma International in 1997 for standardization, which resulted in ECMAScript
Because JavaScript conformed to the ECMAScript specification, JavaScript is an example of an ECMAScript implementation
From 2015 ECMAScript versions are named by year
Not all browsers support all the features in all the versions of ECMAScript

https://www.w3schools.com/js/js_versions.asp

ES5 / ES6 / ES7

Microsoft Visual Studio Code

https://code.visualstudio.com/Download

DIFFERENCES BETWEEN SERVER SIDE AND CLIENT SIDE CODE

Execution of code (Server || Client)

JavaScript is interpreted and executed on the client

Access to source code

JavaScript is sent to the client along with HTML / CSS

HTML5 / CSS3 / JAVASCRIPT

JavaScript is one of the three core technologies of the World Wide Web

- HTML Content / Structure
- CSS Appearance

- JAVASCRIPT Behavior

BASICS

Events / Script tag (Head/Body) / External .js file

JavaScript can be added to a web site in a few different ways

Event

onclick="alert();"

Script tag (Head/Body)

```
<script>
   alert();
</script>
```

Script tag(External .js file)

```
<script src="javascript.js"></script>
```

Debugging

```
alert("");
console.log("");
```

Browser Developer Tools

Inspect / Console

Strict

Indicate that the code should be executed in "strict mode".
With strict mode, you cannot as an example use undeclared variables or objects among other things
Makes it easier to write "secure" JavaScript by not accepting previously accepted "bad syntax"

Beginning of a script or a function

"use strict";

VARIABLES / DATA TYPES

Loosely typed language
Lack of type check
No need to declare variable types explicitely
The type of a variable is the type of its value
Conversions are performed automatically
Type coercion = Conversion between different object types

Var

- Manage values / objects
- Var keyword
- Identifier

  Names can contain letters, digits, underscores, and dollar signs.
  Names must begin with a letter
  Names can also begin with $ and _
  Names are case sensitive
  Reserved words cannot be used as names

Data types

6 primitive data types

String / Boolean / Number / Symbol / Null / Undefined

1 object data type

Declaring / Assigning

```
var carName;                                      // Declaring
carName = "Volvo";                                // Assigning
var carName = "Volvo";                            // Declaring & Assigning
```

Declaring multiple

```
var user = "TomcatManager", appName = "tomcat", price = 500;
```

Redeclaring

```
var city = "Tokyo";
var city;
```

Undefined

Uninitialized properties in JavaScript are not set to null as the default value
Properties without definitions are undefined

```
var person;                                      // Value is undefined, type is undefined
person = undefined;                              // Value is undefined, type is undefined
```

Null

Null values must be set explicitly

```
person = null;
```

Dynamic data types

```
var x;                                            // Now x is undefined
var x = 5;                                        // Now x is a Number
var x = "John";                                   // Now x is a String
```

Missing var

```
fullName = “Donald Duck”;                         // var fullName = “Donald Duck”;
```

Local variables

Inside functions
Deleted when function completes

Global variables / Automatic

Outside functions
Deleted when window closes

Data types

```
var length = 16;                                  // Number
var lastName = "Johnson";                         // String
var cars = ["Saab", "Volvo", "BMW"];              // Array
var somebody = {firstName:"John", lastName:"Doe"}; // Object
```

Typeof / Instanceof

Operators which can be used to check data types of variables

```
var myVar;                           //Declared, but undefined
if (typeof myVar === 'undefined'){ console.log("myVar is undefined"); }

var aVar = {};
if(aVar instanceof Object){ console.log("aVar is an instance of Object"); }
```

Comparisons

Both same value and same type between two expressions can be compared at the same time

```
==              Only value
                            77 == '77'        //true, but not same types: Number == String
===             Both value and type
                            77 === '77'       //false, because not same types: Number === String
                            77 === 77         //true, because same types: Number === Number
```

OBJECTS

W3Schools.com JavaScript Objects Reference

- Boolean / Number / Math / String / Date / Regexp / Array / JSON

```
var length = 16.00;                                      // Number
var lastName = "Johnson";                                // String
var cars = ["Saab", "Volvo", "BMW"];                     // Array
var somebody = {firstName:"John", age:50};               // Object
```

- Objects have properties and methods / functions
- Do not declare strings, numbers, and booleans as objects (Slows down execution speed)

```
var lastName = new String();                             // Declares lastName as a String object
var length = new Number();                               // Declares length as a Number object
var z = new Boolean();                                   // Declares z as a Boolean object
```

Objects are variables containing variables
Objects can contain many different values

Creation
Custom objects can be created with object literal initializers or constructor functions

Object literal initializer

```
var car1 = {color: "red", wheels: 4};
```

Object constructor function

```
function Car(color, wheels) {
   this.color = color;
   this.wheels = wheels;
}
var car2 = new Car("red", 4);
```

Properties
Objects are containers for named values
Name : Value pairs
Access properties and methods / functions via names
Properties of JavaScript objects can be accessed or set using dot notation
Properties of JavaScript objects can also be accessed or set using a bracket notation
Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it

Dot notation

```
console.log(car1.color);
```

Bracket notation

```
console.log(car2["color"]);
```

Methods
Methods are actions that can be performed on objects

Constructor function
When a function is invoked via the new operator, it becomes a constructor function
By convention constructor functions always start with a capital letter
A constructor function is just a function being invoked with new

this
Whenever a function is contained in the global scope, the value of this inside of that function will be the global object (window in a browser) or undefined if in strict mode
Whenever a function is called by a preceding dot, the object before that dot is this.
Whenever a constructor function is used, this refers to the specific instance of the object that is created and returned by the constructor function.

PROTOTYPES

Each constructor function has a prototype property that refers to an object and that object becomes the prototype of all instances created with the constructor function
We can attach new functions and properties to this object, which will be shared by all instances
Every JavaScript object has a prototype object where they inherit properties and methods from

Creating prototypes with constructor function

```
function FamilyMember(first, last, age) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.name = function() {return this.firstName + " " + this.lastName;};
}
var myFather = new FamilyMember("John", "Doe", 50);
var myMother = new FamilyMember("Jane", "Doe", 48);
```

Adding properties and methods

```
FamilyMember.prototype.nationality = "English";
FamilyMember.prototype.nameUpper = function(){return (this.firstName + " " + this.lastName).toUpperCase()};
```

Accessing protoype property and method

```
myFather.firstName);
console.log(myMother["lastName"]);
```

ARRAYS

Methods

ForEach / Concat / Join / Unshift / Push / Shift / Pop / Splice / Reverse / Sort / Map / Filter / Reduce

CONTROL STRUCTURES

Selection

if / else if / else / switch

Iteration

for / while / do while

FUNCTIONS

Block of code to perform a particular task

```
function myFunction(p1, p2) {
   return p1 * p2;
}
```

Executes when invoked

```
myFunction(100, 300);
```

Parameters / Arguments

Arguments object / Arguments array

```
function someFunction(p1, p2) {
   return arguments[0] * arguments[1];
}
console.log(someFunction(100, 300));
```

Anonymous functions

Dynamically declared at runtime
Declared without named identifier
Can be used as parameter or stored in variable and invoked with variable name

```
var sub = function(n1, n2){
  return n1 - n2;
}
console.log(sub(8,2));

function doIt(anonymous)
{
  anonymous();
}
doIt(function(){console.log("Anonymous function...")});
```

Self invoking functions

Wrap anonymous function

Runs immediately

```
(function(){
  console.log("SelfInvoked Syntax1!!!");
})();

!function(){
  console.log("SelfInvoked Syntax2!!!");
}()
```

Functions are first class members

Functions as variables

```
var f1 = function(){};
```

Functions as parameters

```
var f2 = strangeFunction(f1);
```

Functions as returns

```
function strangeFunction(p1){
  return function() {console.log("Returning function...")};
}
```

Function callbacks

A callback is a function that is to be executed after another function (normally asynchronous) has finished executing

```
function simpleFunction(p1, p2, callback)
{
  console.log('The parameters: ' + p1 + ', ' + p2);

  callback();
}
simpleFunction(3,5,function(){ console.log("Do this...")});
simpleFunction(3,5,function(){ console.log("Do something else...")});
```

Asynchronous callbacks

```
function aAsync(){
  setTimeout(function(){ console.log("Delayed..."); },2000);
}
aAsync();
console.log("What come first, this or delayed...");
```

Synchronous callbacks

```
var numbers = [1, -4, 9];
var newSign = numbers.map(function(num)
{
  return num * -1;
});
console.log(numbers);
console.log(newSign);
```

Array callbacks

```
var names = ["kurt","ole","hans","ib"];
names.forEach(function(name){
    console.log(name);
});
var newArray = names.filter(function(name){
    return name.length <= 3;
});
console.log(newArray);
var mapArray = names.map(function(name){
    return name.toUpperCase();
});
console.log(mapArray);
```

Nested functions
Functions only available inside surrounding function
Not within loops or conditionals

```
function containerFunction()
{
    function NestedFunction()
    {
        console.log("NestedFunction…");
    };
    NestedFunction();
}
containerFunction();
```

Closures
A closure is a special kind of object that combines two things:

- A function
- The environment in which that function was created. The environment consists of any local variables that were in-scope at the time that the closure was created

Private variables can be created with closures
it is possible to emulate private methods using closures
Using closures in this way is known as the module pattern
Functions that refer to variables that are used locally, but defined in an enclosing scope
Functions 'remember' the environment in which they were created
Nested functions become global
Inner function is made accessible from outside of the function that created it
Variables can only be changed by nested functions

```
var makeCounter = function() {
    var privateCounter = 0;
    function changeBy(val) {
     privateCounter += val;
    }
    return {
     increment: function() {changeBy(1);},
     decrement: function() {changeBy(-1);},
     value: function() { return privateCounter;}
    }
};
var counter1 = makeCounter();
var counter2 = makeCounter();
counter1.increment();
counter1.increment();
console.log(counter1.value());
console.log(counter2.value());
```

HOISTING

Variables / Functions

JavaScript does not support block scoping.

Variable definitions are not scoped to their nearest enclosing statement or block, as in Java, but rather to their containing function.

All declarations, both functions and variables, are hoisted to the top of the containing scope, before any part of your code is executed
Functions are hoisted first, and then variables are hoisted
Variable hoisting / Problems

```
var x = 10;
function y() {
   alert(x);            //Prints undefined
   var x = 20;
   alert(x);            //Prints 20
}
```

Function hoisting

```
myFunction(5);
function myFunction(a, b) {
   return a * b;
}
myFunction(5);
```