

JAVASCRIPT

Interpreted programming language

Core technology of the World Wide Web

Also used in many other situations

ECMAScript

JavaScript engines

Server side <-> Client side

Execution / Source code

EDITOR

Microsoft Visual Studio Code

<https://code.visualstudio.com/Download>

JAVASCRIPT

Web programming

- HTML - Content / Structure
- CSS - Appearance
- JAVASCRIPT - Behavior

JAVASCRIPT / HTML

Events / Script tag

- Events
- Script tag (head / body)
- Script tag (src=External .js file)

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript</title>
  </head>
  <body>
    <h1>Javascript</h1>
    <script src="Javascript.js"></script>
  </body>
</html>
```

Variables

Loosely typed language

Lack of type check

No need to declare variable types explicitly

The type of a variable is the type of its value

Conversions are performed automatically

Type coercion = Conversion between different object types

Variables

- 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

- Declaring / Assigning / Both

```
var carName;  
carName = "Volvo";  
var carName = "Volvo";
```

- Declaring multiple

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

- Redeclaring

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

- Undefined

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

Variables

- Null

person = null;

- Dynamic data types

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

- Missing var

X = "Donald Duck";

- Local variables

Inside functions

Deleted when function completes

- Global variables

Outside functions

Deleted when window closes

Variables

■ Data types

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

■ Typeof operator

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

■ Instanceof operator

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

■ Comparisons

```
77 == '77' //true, but not same types: Number == String
77 === '77' //false, because not same types: Number === String
77 === 77 //true, because same types: Number === Number
```

Data types / Objects

- 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

```
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
```

- W3Schools.com Javascript Objects Reference

<https://www.w3schools.com/jsref/default.asp>

Objects

- Objects are containers for named values
- Name : Value pairs
- Access properties and methods / functions via names
- Objects are Variables Containing Variables
- Objects can contain many values
- Custom objects can be created with object literals or constructor functions
- Prototypes

Prototypes

- 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 prototype property and method

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

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));
```

- Return statement
- Reuse code / Different arguments

Functions

- Anonymous functions

Declared without named identifier

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("I am anonymous...")});
```


Functions

- Self invoking functions

Wrap anonymous function

Runs immediately

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

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

Functions

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...")};  
}
```


Functions

- Callbacks

Function is given as parameter and executed before function finishes

```
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...")});
```

Functions

- Callbacks

Function is given as parameter and executed before function finishes

```
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...")});
```

Functions

- Asynchronous

Asynchronous functions will not wait for each other to execute

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

- Synchronous

Synchronous functions will wait for each other to execute

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

Functions

- Nested functions

Functions only available inside surrounding function

Not within loops or conditionals

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

- Closure

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

*Makes it possible for a function to have "**private**" variables*

Inner function is made accessible from outside of the function that created it

Variables can only be changed by nested functions

Functions

- Closure example

```
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

- 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() {  
    console.log(x);  
    var x = 20;  
    console.log(x);  
}  
y();
```

- Function hoisting

```
hoistedFunction();  
function hoistedFunction(a, b) {  
    console.log("HoistedFunction...");  
}  
hoistedFunction();
```

DOM Objects

Document / Element / Style / innerHTML / innerText / Attributes / Value / Events

- getElementById / getElementsByTagName / getElementsByClassName
- querySelector
- Style
- innerHTML / innerText
- setAttribute
- value

DOM Objects / Element Objects

- DOM Objects

Attributes Console Document Elements Events Event Objects Geolocation History HTMLCollection Location Navigator Screen Style Window Storage

- Element Objects

a abbr address area article aside audio b base bdo blockquote body br button canvas caption cite code col colgroup datalist dd del details dfn dialog div dl dt em embed fieldset figcaption figure footer form head header h1 - h6 hr html i iframe img ins input button input checkbox input color input date input datetime input datetime-local input email input file input hidden input image input month input number input password input radio input range input reset input search input submit input text input time input url input week kbd label legend li link map mark menu menuitem meta meter nav object ol optgroup option output p param pre progress q s samp script section select small source span strong style sub summary sup table td th tr textarea time title track u ul var video

- W3Schools.com Javascript DOM Reference

JSON - Intro

JavaScript Object Notation

- JSON stands for **JavaScript Object Notation**
- JSON is a lightweight data-interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand
- JSON evaluates to JavaScript Object

```
var text = '{"name":"John Johnson","street":"Oslo West 16","phone":"555 1234567"}';  
var obj = JSON.parse(text);  
document.getElementById("Person").innerHTML = obj.name;
```

JSONObjects.json

```
{"employees": [  
  {"firstName": "John", "lastName": "Doe"},  
  {"firstName": "Anna", "lastName": "Smith"},  
  {"firstName": "Peter", "lastName": "Jones"}  
]}
```

JSON Rules

Syntax rules

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

FETCH

- Basic fetch

```
fetch(url)
  .then(function() {
  })
  .catch(function() {
  });
```

- Fetch example

```
fetch("https://reqres.in/api/users/2")
  .then(function(response){
    if (response.status !== 200)
    {
      console.log("Problem occurred... Status Code: " + response.status);
      return;
    }

    response.json().then(function(data)
    {
      console.log(data);
      console.log(data.data.avatar);
      document.getElementById("imgJson").setAttribute("src", data.data.avatar);
    });
  })
  .catch(function(error) {
    console.log("Error occurred...", error);
  });
```