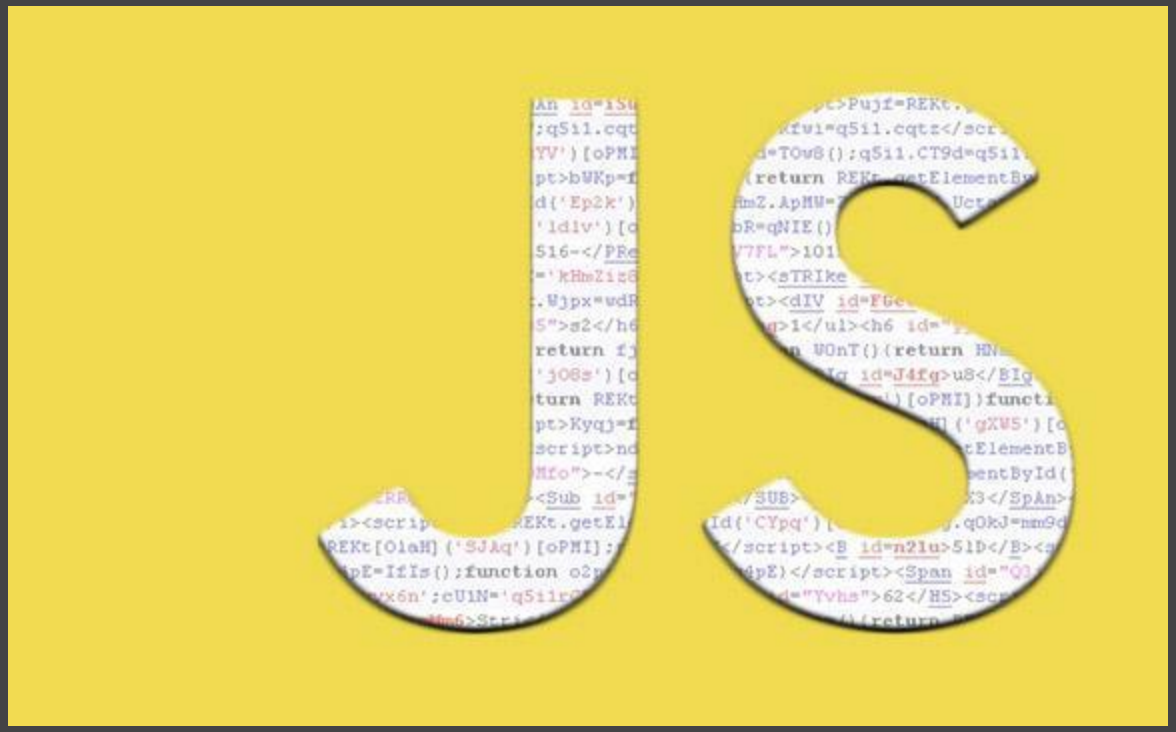
Introduction to JavaScript


JS

What is JavaScript ?

JavaScript is a lightweight, interpreted programming language.

What is JavaScript ?

JavaScript is a lightweight, interpreted programming language.

Advantages

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces

What is JavaScript ?

JavaScript is a lightweight, interpreted programming language.

Advantages

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces

Limitations

- Client-side JavaScript does not allow the reading or writing of files.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

JavaScript Where To

The <script> Tag

```
<script>
   document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript Where To

The <script> Tag

```
<script>
    document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript in <head>

```
<head>
    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = "Paragraph changed.";
        }
    </script>
</head>
```

JavaScript Where To

The <script> Tag

```
<script>
    document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript in <head>

```
<head>
    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = "Paragraph changed.";
        }
    </script>
</head>
```

JavaScript in <body>

```
<body>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
<script>
    function myFunction() {
        document.getElementById("demo").innerHTML = "Paragraph changed.";
    }
</script>
</body>
```

External JavaScript

```
<!DOCTYPE html>
<html>
<body>
<script src="my-script.js"></script>
</body>
</html>
```

External JavaScript

```
<!DOCTYPE html>
<html>
<body>
<script src="my-script.js"></script>
</body>
</html>
```

Advantages:

- It separates HTML and code.
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads.

JavaScript Syntax

Comments:

```
// var x = 5 + 6; I will not be executed

var x = 5;        // Declare x, give it the value of 5

/*
Not all JavaScript statements are "executable commands".
Anything after double slashes // is treated as a comment.
Comments are ignored, and will not be executed:
*/
```

All JavaScript identifiers are **case sensitive**.

```
lastName != lastname
```

JavaScript uses the **Unicode** character set.

JavaScript Display Possibilities

- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.
- Writing into an HTML element, using **innerHTML**.

JavaScript Display Possibilities

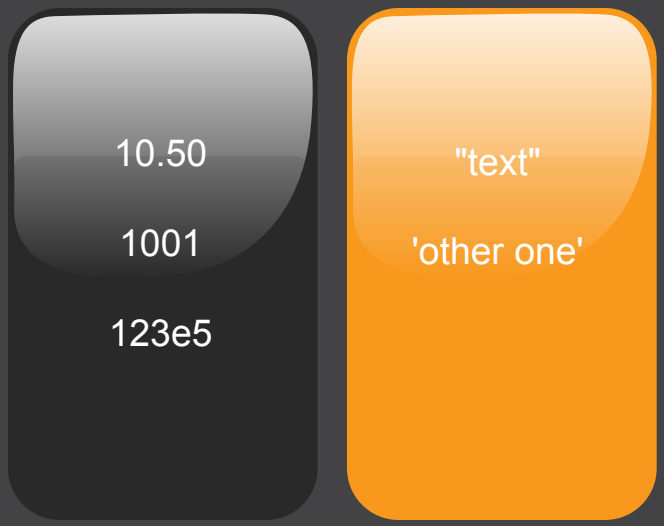
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.
- Writing into an HTML element, using **innerHTML**.

```
document.write(5 + 6);

document.getElementById("demo").innerHTML = 5 + 6;

window.alert(5 + 6);

console.log(5 + 6);
```

JavaScript Statements

Values

Numbers are written with or without decimals



Strings are written with double or single quotes

Values

Numbers are written with or without decimals

```
10.50
1001
123e5
```

Strings are written with double or single quotes

```
"text"
'other one'
```

Variables

Variables are used to **store** values

```
var x;
x = 6;
```

```
var y = 5;
var z = y + 1;
```

The **var** keyword to **define** variables

JavaScript Identifiers

All JavaScript **variables** (and JavaScript functions) must be **identified** with **unique names**.

The general rules for constructing a names for variables (unique identifiers) are:

- Names must begin with a letter
- Names can also begin with $ and _
- Names can contain letters, digits, underscores, and dollar signs.
- Names are case sensitive (y and Y are different variables)
- Can be used “camelCase”
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript Identifiers

All JavaScript **variables** (and JavaScript functions) must be **identified** with **unique names**.

The general rules for constructing a names for variables (unique identifiers) are:

- Names must begin with a letter
- Names can also begin with $ and _
- Names can contain letters, digits, underscores, and dollar signs.
- Names are case sensitive (y and Y are different variables)
- Can be used “camelCase”
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript variables can hold many types of data.

JavaScript Identifiers

All JavaScript **variables** (and JavaScript functions) must be **identified** with **unique names**.

The general rules for constructing a names for variables (unique identifiers) are:

- Names must begin with a letter
- Names can also begin with $ and _
- Names can contain letters, digits, underscores, and dollar signs.
- Names are case sensitive (y and Y are different variables)
- Can be used “camelCase”
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript variables can hold many types of data.

In JavaScript, the equal sign (=) is an "assignment" operator, is not an "equal to" operator.

```
x = x + 5
```

JavaScript Identifiers

All JavaScript **variables** (and JavaScript functions) must be **identified** with **unique names**.

The general rules for constructing a names for variables (unique identifiers) are:

- Names must begin with a letter
- Names can also begin with $ and _
- Names can contain letters, digits, underscores, and dollar signs.
- Names are case sensitive (y and Y are different variables)
- Can be used “camelCase”
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript variables can hold many types of data.

In JavaScript, the equal sign (=) is an "assignment" operator, is not an "equal to" operator.

```
x = x + 5
```

If you put quotes around a numeric value, it will be treated as a text string.

```
var pi = '3.14';
var constPi=3.14;
```

Declaring (Creating) JavaScript Variables

```
var carName;
//Variable declared without a value will have the value undefined.
carName = "Volvo";

//or

var carName = "Volvo";

//For many Variables start the statement with var and separate the
variables by comma:
var lastName = "Doe",
   age = 30,
   job = "carpenter";

//or

var lastName = "Doe";
var age = 30;
var job = "carpenter";
```

Global variables are evil!!!!

```
evil = 'Variable declaration without var'
```

JavaScript Keywords
(reserved words)

break	Terminates a switch or a loop
catch	Marks the block of statements to be executed when an error occurs in a try block
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
for ... in	Marks a block of statements to be executed, for each element of an object (or array)
function	Declares a function

JavaScript Keywords
(reserved words)

if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
throw	Throws (generates) an error
try	Implements error handling to a block of statements
var	Declares a variable
while	Marks a block of statements to be executed, while a condition is true

JavaScript Data Types

```
var albums = 16;                                    // Number assigned by a number literal
var songs = albums * 10;                            // Number assigned by an expression literal
var title = "Highway to Hell";                      // String assigned by a string literal
var members = [
    "Angus Young",
    "Phil Rudd",
    "Cliff Williams",
    "Brian Johnson",
    "Stevie Young"
];                                                  // Array  assigned by an array literal
var band = {name:"AC/DC", startYear:1973};          // Object assigned by an object literal
```

JavaScript Data Types

Number:
Integer
Float
Infinity
NaN

String:
“text1”
‘text2’

Boolean:
true
false

Object:
object
null
Array
Function

undefined

The typeof Operator

```
typeof "Cat"
```

The typeof Operator

```
typeof "Cat"                    // Returns string
typeof 3.14
```

The typeof Operator

```
typeof "Cat"                    // Returns string
typeof 3.14                     // Returns number
typeof false
```

The typeof Operator

```
typeof "Cat"                    // Returns string
typeof 3.14                     // Returns number
typeof false                    // Returns boolean
typeof [1,2,3,4]
```

The typeof Operator

```
typeof "Cat"                    // Returns string
typeof 3.14                     // Returns number
typeof false                    // Returns boolean
typeof [1,2,3,4]                // Returns object
typeof {name:'Ann', age:17}
```

The typeof Operator

```
typeof "Cat"                  // Returns string
typeof 3.14                   // Returns number
typeof false                  // Returns boolean
typeof [1,2,3,4]              // Returns object
typeof {name:'Ann', age:17}   // Returns object
```

```
<!DOCTYPE html>
<html>
<body>
<p>The typeof operator returns the type of a variable or an
expression.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
   typeof "Cat" + "<br>" +
   typeof 3.14 + "<br>" +
   typeof false + "<br>" +
   typeof [1,2,3,4] + "<br>" +
   typeof {name:'Ann', age:17};
</script>
</body>
</html>
```

Strings

Length:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;
var sln = txt.length;
```

Strings

Length:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;
var sln = txt.length;
```

Special Characters:

*** escape character **

\'	single quote
\"	double quote
****	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	page break

Strings

Created from literals:

```
var x = "Lorem ipsum";
```

Strings

Created from literals:

```
var x = "Lorem ipsum";
```

Defined as objects:

```
var y = new String("Lorem ipsum");
```

Strings

Created from literals:

```
var x = "Lorem ipsum";
```

Defined as objects:

```
var y = new String("Lorem ipsum");
```

```
console.log(typeof x); //?
console.log(typeof y); //?
console.log(x === y); //?
```

Strings

Created from literals:

```
var x = "Lorem ipsum";
```

Defined as objects:

```
var y = new String("Lorem ipsum");
```

```
console.log(typeof x); //?
console.log(typeof y); //?
console.log(x === y); //?
```

Don't create String objects. They slow down execution speed!

String Methods

charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings

String Methods

substr()	Extracts a part of a string from a start position through a number of characters
substring()	Extracts a part of a string between two specified positions
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

String Methods

Finding a String in a String:

indexOf()

```
var str = "The indexOf() method returns the index of (the position of) the first
occurrence of a specified text in a string";
var pos = str.indexOf("index");
```

String Methods

Finding a String in a String:

indexOf()

```
var str = "The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string";
var pos = str.indexOf("index");
```

lastIndexOf()

```
var str = "The lastIndexOf() method returns the index of the last occurrence of a specified text in a string";
var pos = str.lastIndexOf("index");
```

String Methods

Finding a String in a String:

indexOf()

```
var str = "The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string";
var pos = str.indexOf("index");
```

lastIndexOf()

```
var str = "The lastIndexOf() method returns the index of the last occurrence of a specified text in a string";
var pos = str.lastIndexOf("index");
```

Return -1 if the text is not found!
JavaScript counts positions from zero.
Both methods accept a second parameter as the starting position for the search.

String Methods

Finding a String in a String:

indexOf()

```
var str = "The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string";
var pos = str.indexOf("index");
```

lastIndexOf()

```
var str = "The lastIndexOf() method returns the index of the last occurrence of a specified text in a string";
var pos = str.lastIndexOf("index");
```

Return -1 if the text is not found!
JavaScript counts positions from zero.
Both methods accept a second parameter as the starting position for the search.

search()

```
var str = "The search() can work with regular expressions and returns the position of the match";
var pos = str.search("search");
```

String Methods

Extracting String Parts:

slice(start [, end])

```
var str = "The method takes 2 parameters: the starting index (position), and the ending index (position).";
var res = str.slice(4,10);
```

String Methods

Extracting String Parts:

slice(start [, end])

```
var str = "The method takes 2 parameters: the starting index (position), and the ending index (position).";
var res = str.slice(4,10);
```

or

```
var str = "If a parameter is negative, the position is counted from the end of the string.";
var res = str.slice(-7,-1);
```

String Methods

Extracting String Parts:

slice(start [, end])

```
var str = "The method takes 2 parameters: the starting index (position), and the ending index (position).";
var res = str.slice(4,10);
```

or

```
var str = "If a parameter is negative, the position is counted from the end of the string.";
var res = str.slice(-7,-1);
```

Try with one parameter!!!

String Methods

Replacing String Content:

replace(regexp|substr, newSubStr|function)

```
str = "One coffee, please!";
var n = str.replace("coffee","tea");
```

String Methods

Replacing String Content:

replace(regexp|substr, newSubStr|function)

```
str = "One coffee, please!";
var n = str.replace("coffee","tea");
```

Converting to Upper and Lower Case:

```
var text1 = "Hello World!";
var text2 = text1.toUpperCase();
var text3 = text1.toLowerCase();
```

String Methods

concat(*string1, string2, ..., stringX*)

The **concat()** method can be used instead of the plus operator.

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ","World!");
```

Converting a String to an Array:

split(reg|substr)

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
var txt = "a bb c d e";   // String
txt.split(" ");          // Split on spaces
var txt = "a|b|c|d|e";   // String
txt.split("|");          // Split on pipe
```

Numbers

By default, Javascript displays numbers as base 10 decimals.

Base 16 (hex), base 8 (octal), or base 2 (binary) can be used.

```
var myNumber = 128;
myNumber.toString(16);      // returns 80
myNumber.toString(8);       // returns 200
myNumber.toString(2);       // returns 10000000
```

Numbers

Infinity

```
// Execute until Infinity
<script>
function myFunction() {
    var myNumber = 2;
    var txt = "";
    while (myNumber != Infinity) {
        myNumber = myNumber * myNumber;
        txt = txt + myNumber + "<br>";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script>
```

or

```
var x =  2 / 0;
```

Numbers

Infinity

```
// Execute until Infinity
<script>
function myFunction() {
    var myNumber = 2;
    var txt = "";
    while (myNumber != Infinity) {
        myNumber = myNumber * myNumber;
        txt = txt + myNumber + "<br>";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script>
```

or

```
var x =  2 / 0;
```

NaN - Not a Number

```
var x = 100 / "Apple";
var x = 100 / "10";
isNaN(x); // returns true because x is Not a Number
```

Numbers

Created from literals:

```
var x = 123;
```

Numbers

Created from literals:

```
var x = 123;
```

Defined as objects:

```
var y = new Number(123);
```

Numbers

Created from literals:

```
var x = 123;
```

Defined as objects:

```
var y = new Number(123);


typeof x;   //?
typeof y;   //?
(x === y)   //?
```

Numbers

Created from literals:

```
var x = 123;
```

Defined as objects:

```
var y = new Number(123);
```

```
typeof x;   //?
typeof y;   //?
(x === y)   //?
```

Don't create Number objects. They slow down execution speed!

Global methods for numbers

Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns a floating point number

Try:

```
x = true;
Number(x);
x = false;
Number(x);
x = new Date();
Number(x);
x = "10"
Number(x);
x = "10 20"
Number(x);
```

```
parseInt("10");
parseInt("10.33");
parseInt("10 20 30");
parseInt("10 years");
parseInt("years 10");
```

```
parseFloat("10");
parseFloat("10.33");
parseFloat("10 20 30");
parseFloat("10 years");
parseFloat("years 10");
```

Number methods

toString()

```
var x = 123;

x.toString();            // returns 123 from variable x

(123).toString();        // returns 123 from literal 123

(100 + 23).toString();   // returns 123 from expression 100 + 23
```

Number methods

toString()

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
```

toFixed(precision)

returns a string, with the number written with a specified number of decimals

```
var x = 9.656;
x.toFixed(0);            // returns 10
x.toFixed(2);            // returns 9.66
x.toFixed(4);            // returns 9.6560
```

Number methods

toString()

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
```

toFixed(precision)

returns a string, with the number written with a specified number of decimals

```
var x = 9.656;
x.toFixed(0);            // returns 10
x.toFixed(2);            // returns 9.66
x.toFixed(4);            // returns 9.6560
```

toPrecision(precision)

returns a string, with a number written with a specified length

```
var x = 9.656;
x.toPrecision();         // returns 9.656
x.toPrecision(2);        // returns 9.7
x.toPrecision(4);        // returns 9.656
```

Booleans

Everything With a Real Value is True:

```
Boolean(100);
Boolean(3.14);
Boolean(-15);
Boolean("Hello");
Boolean('false');
Boolean(1 + 7 + 3.14);
```

Booleans

Everything With a Real Value is True:

```
Boolean(100);
Boolean(3.14);
Boolean(-15);
Boolean("Hello");
Boolean('false');
Boolean(1 + 7 + 3.14);
```

Everything Without a Real Value is False:

```
Boolean(0);
Boolean(-0);
Boolean(x);
Boolean("");
Boolean(null);
Boolean(false);
Boolean(10 / "A");
```

Arrays

JavaScript arrays are used to store multiple values in a single variable.

```
var array-name = [item1, item2, ...];

var cars = ["Saab", "Volvo", "BMW"];
var cars = new Array("Saab", "Volvo", "BMW");
```

Arrays

JavaScript arrays are used to store multiple values in a single variable.

```
var array-name = [item1, item2, ...];
```

```
var cars = ["Saab", "Volvo", "BMW"];
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array:

```
var name = cars[0];
cars[0] = "Opel";
```

[0] is the first element in an array. [1] is the second. Array indexes start with 0.
In JavaScript, **arrays** use **numbered indexes**.

Arrays

JavaScript arrays are used to store multiple values in a single variable.

```
var array-name = [item1, item2, ...];
```

```
var cars = ["Saab", "Volvo", "BMW"];
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array:

```
var name = cars[0];
cars[0] = "Opel";
```

[0] is the first element in an array. [1] is the second. Array indexes start with 0.
In JavaScript, **arrays** use **numbered indexes**.

Looping Array Elements

```
var index,
   text="";
var fruits = ["Banana", "Orange", "Apple", "Mango"];
for    (index = 0; index < fruits.length; index++) {
   text += fruits[index];
}
console.log(text);
```

Arrays

Avoid new Array()!

Try:

```
var points = new Array(40, 100);
```

```
var points = new Array(40);
```

Arrays

Avoid new Array()!

Try:

```
var points = new Array(40, 100);
var points = new Array(40);
```

How to Recognize an Array?

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
typeof fruits;
```

Arrays

Avoid new Array()!

Try:

```
var points = new Array(40, 100);
var points = new Array(40);
```

How to Recognize an Array?

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
typeof fruits;
```

better

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = isArray(fruits);
function isArray(myArray) {
   return myArray.constructor.toString().indexOf("Array") > -1;
}
</script>
```

Array Methods

Converting Arrays to Strings

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

valueOf()

```
document.getElementById("demo").innerHTML = fruits.valueOf();
```

Array Methods

Converting Arrays to Strings

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

valueOf()

```
document.getElementById("demo").innerHTML = fruits.valueOf();
```

toString()

```
document.getElementById("demo").innerHTML = fruits.toString();
```

Array Methods

Converting Arrays to Strings

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

valueOf()

```
document.getElementById("demo").innerHTML = fruits.valueOf();
```

toString()

```
document.getElementById("demo").innerHTML = fruits.toString();
```

join(str)

```
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Array Methods

Add and remove elements

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
```

pop() removes the last element from an array

```
fruits.pop();
```

Array Methods

Add and remove elements

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
```

pop() removes the last element from an array

```
fruits.pop();
```

push() method adds a new element to an array (at the end)

```
fruits.push("Kiwi");
```

Array Methods

Add and remove elements

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
```

pop() removes the last element from an array

```
fruits.pop();
```

push() method adds a new element to an array (at the end)

```
fruits.push("Kiwi");
```

shift() removes the first element of an array, and "shifts" all other elements one place down

```
fruits.shift();
```

Array Methods

Add and remove elements

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
```

pop() removes the last element from an array

```
fruits.pop();
```

push() method adds a new element to an array (at the end)

```
fruits.push("Kiwi");
```

shift() removes the first element of an array, and "shifts" all other elements one place down

```
fruits.shift();
```

unshift() adds a new element to an array (at the beginning), and "unshifts" older elements

```
fruits.unshift("Lemon");
```

The shift() method returns the string that was "shifted out".

The unshift() method returns the new array length.

Array Methods

Add and remove elements

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
```

splice(index[, deleteCount, elem1, ..., elemN])

```
fruits.splice(2, 0, "Lemon", "Kiwi");
```

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

Array Methods

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
```

sort()

```
fruits.sort();
```

reverse()

```
fruits.reverse();
```

Objects

Hobbit

Objects

Hobbit

Hobbit.name = “Bilbo”
Hobbit.age = 132
Hobbit.address = Shire
Hobbit.hair = true

Hobbit.walk()
Hobbit.fight()
Hobbit.keepRing()

Objects

Hobbit

Hobbit.name = “Bilbo”
Hobbit.age = 132
Hobbit.address = Shire
Hobbit.hair = true

Hobbit.walk()
Hobbit.fight()
Hobbit.keepRing()

Gollum

Objects

Hobbit

Hobbit.name = “Bilbo”
Hobbit.age = 132
Hobbit.address = Shire
Hobbit.hair = true

Hobbit.walk()
Hobbit.fight()
Hobbit.keepRing()

Gollum

gollum = new Hobbit();
gollum.name = “Gollum”
gollum.age = null
gollum.address = cave
gollum.hair = false

gollum.eatFreshFish()

Objects

All hobbits have the same **properties**, but the property values differ from one to one.

All hobbits have the same **methods**, but the methods are performed at different times.

```
var person = {
    firstName:"Bilbo", //property
    lastName:"Baggins", //property
    age:132, //property
    eyeColor:"blue", //property
    walk: function(){
      console.log('walking to Mordor'); //method
    }
};
```

Objects

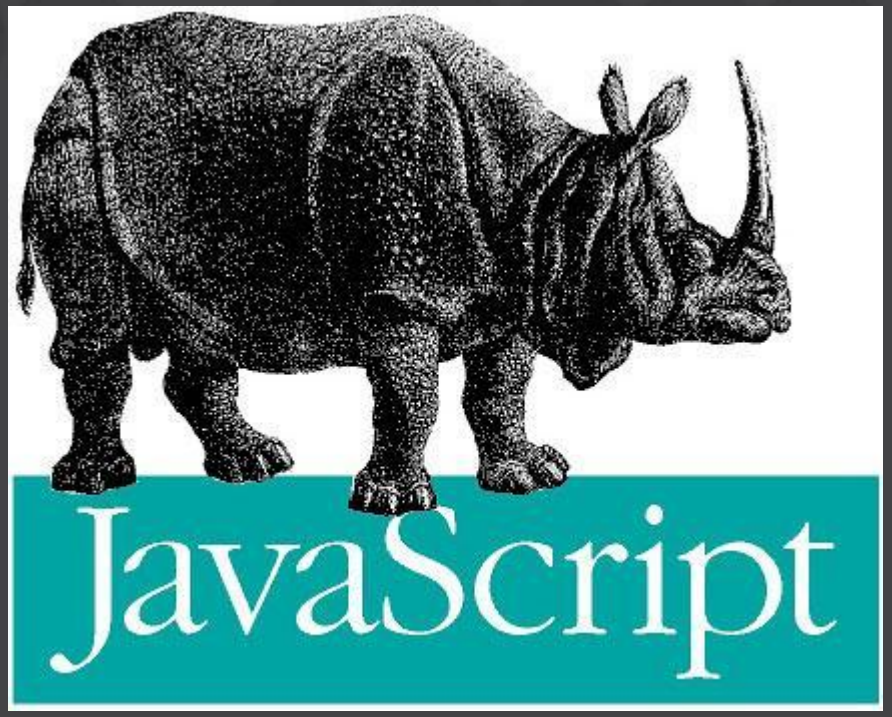
All hobbits have the same **properties**, but the property values differ from one to one.

All hobbits have the same **methods**, but the methods are performed at different times.

```
var person = {
    firstName:"Bilbo", //property
    lastName:"Baggins", //property
    age:132, //property
    eyeColor:"blue", //property
    walk: function(){
      console.log('walking to Mordor'); //method
    }
};

person.address = "Shire";
person.fight = function(){
    console.log('blood');
};
```

Objects

```
var x = lastName;
console.log(person.firstName);
console.log(person[x]);
person.walk();
```



You are a hero if you'll read this book!