

## What is JavaScript?

JavaScript is a scripting language that is created for showing HTML pages live. JavaScript does not require any compilation. It is interpreted language. JavaScript can modify HTML page, write text in it, add or remove tags, change styles etc.

JavaScript code can get to execute on events, mouse clicks, and movements, keyboard input etc. It can send the request to the server and load data without reloading of the page. This technology is often called “AJAX”. JavaScript makes the web pages dynamic.

## Advantages of JavaScript

- Adding new HTML to the page, change the existing content, modify styles.
- React to the user actions, run on mouse clicks, pointer movements, key presses.
- Send requests through the network to remote servers, download and upload files. It is called AJAX and COMET Technologies.

In this **JavaScript tutorial**, we will learn different JavaScript Features and how we inculcate it in writing HTML pages. It gives insight about how modern features of JavaScript are widely used.

We will discuss different unique features of JavaScript and various features of Modern JavaScript with examples.

## Core Features of JavaScript

Various features of JavaScript are as follows:

### 1. Client – Side Technology

JavaScript is a client-side technology. It is mainly used for giving client-side validation. It is an object-based scripting language.

### 2. Greater Control

It gives the user more control over the browser.

**Example** – You can change the background color of the page as well as text on the browser’s status bar.

Here, the back button is implemented with JavaScript. Click it and you will return to the page from which you have arrived.

### 3. Detecting the User’s Browser and OS.

The feature to detect the user’s browser and OS enables your script to perform platform-dependent operations.

#### 4. Performing Simple Calculation on the Client side

Using a JavaScript calculator, we perform simple calculations on the client side.

#### 5. Validating The User's Input

In the JavaScript calculator, try to type some letters instead of numeric input, you will get an error: Invalid input character. Note that, JavaScript helps the browser perform output validation without wasting the user's time by the web server access.

If the user makes a mistake in the input, the user will get an error message immediately. If the input information is validated only on the server, then the user would have to wait for the server's response.

#### 6. Handling Date and time

The JavaScript says, "Nice Morning, isn't it? Or "Good Afternoon", depending on the current time. It also tells you today's date.

Nice Morning isn't it? Today is Wednesday, 20 February 2019.

#### 7. Generating HTML on the Fly

Every time you click on the button, the browser generates and displays a new HTML code. Thanks to JavaScript, this operation performs on the client- machine and therefore you don't have to wait while the information backs and forth between your browser and the web server.

#### 8. Semicolon Insertion

All statements in JavaScript must terminate with a semicolon. Most of the JavaScript control statements syntax is the same as control statements in C.

#### JavaScript Objects

A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

#### Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal

2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

### 1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

1. `object={property1:value1,property2:value2.....propertyN:valueN}`

As you can see, property and value is separated by : (colon).

Let's see the simple example of creating object in JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

### 2) By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.

Let's see the example of creating object directly.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

### 3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

*Output of the above example*

103 Vimal Jaiswal 30000

---

### Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;

this.changeSalary=changeSalary;
function changeSalary(otherSalary){
this.salary=otherSalary;
}
}
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script>
```

## JavaScript Object Methods

The various methods of Object are as follows:

S.No	Methods	Description
1	<u>Object.assign()</u>	This method is used to copy enumerable and own properties from a source object to a target object
2	<u>Object.create()</u>	This method is used to create a new object with the specified prototype object and properties.
3	<u>Object.defineProperty()</u>	This method is used to describe some behavioral attributes of the property.
4	<u>Object.defineProperties()</u>	This method is used to create or configure multiple object properties.
5	<u>Object.entries()</u>	This method returns an array with arrays of the key, value pairs.
6	<u>Object.freeze()</u>	This method prevents existing properties from being removed.
7	<u>Object.getOwnPropertyDescriptor()</u>	This method returns a property descriptor for the specified property of the specified object.
8	<u>Object.getOwnPropertyDescriptors()</u>	This method returns all own property descriptors of a given object.

9	<u>Object.getOwnPropertyNames()</u>	This method returns an array of all properties (enumerable or not) found.
10	<u>Object.getOwnPropertySymbols()</u>	This method returns an array of all own symbol key properties.
11	<u>Object.getPrototypeOf()</u>	This method returns the prototype of the specified object.
12	<u>Object.is()</u>	This method determines whether two values are the same value.
13	<u>Object.isExtensible()</u>	This method determines if an object is extensible
14	<u>Object.isFrozen()</u>	This method determines if an object was frozen.
15	<u>Object.isSealed()</u>	This method determines if an object is sealed.
16	<u>Object.keys()</u>	This method returns an array of a given object's own property names.
17	<u>Object.preventExtensions()</u>	This method is used to prevent any extensions of an object.
18	<u>Object.seal()</u>	This method prevents new properties from being added and marks all existing properties as non-configurable.
19	<u>Object.setPrototypeOf()</u>	This method sets the prototype of a specified object to another object.

20	<u><a href="#">Object.values()</a></u>	This method returns an array of values.
----	----------------------------------------	-----------------------------------------

## JavaScript Operators with Example

JavaScript Operators use either value or variable to compute some task. This lesson describes the JavaScript operators with example, and operators precedence. JavaScript has following types operators,

1. [Arithmetic Operators](#)
2. [Assignment Operators](#)
3. [Comparison Operators](#)
4. [Logical Operators](#)
5. [Conditional Operator \(Ternary Operator\)](#)
6. [Bitwise Operators](#)
7. [Miscellaneous Operators](#)
  1. [typeof](#)
  2. [delete](#)
  3. [instanceof](#)
  4. [new](#)
  5. [this](#)
  6. [in](#)

## JavaScript Operators with Example

### JavaScript Arithmetic Operators

JavaScript arithmetic operator take operand (as a values or variable) and return the single value.

We are use in our routine life arithmetic operators, addition(+), subtraction(-), multiplication (\*), and division (/) and some other arithmetic operator are listed below.

We have numeric variable: `x = 10, y = 5` and result.

Operator	Description	Example	Results
+	Addition	<code>result = x + y</code>	<code>result = 15</code>
-	Subtraction	<code>result = x - y</code>	<code>result = 5</code>

*	Multiplication	result = x * y	result = 50
/	Division	result = x / y	result = 2
%	Modulus	result = x % y	result = 0
++	Increment	result = x++ result = x result = ++x	result = 10 result = 11 result = 12
--	Decrement	result = x-- result = x result = --x	result = 12 result = 11 result = 10

```
<script>
var x = 10, y = 5;

document.writeln(x + y);      // Addition: 15
document.writeln(x - y);      // Subtraction: 5
document.writeln(x * y);      // Multiplication: 50
document.writeln(x / y);      // Division: 2
document.writeln(x % y);      // Modulus: 0

document.writeln(x++);        // x: 10, x become now 11
document.writeln(x);          // x: 11
document.writeln(++x);        // x become now 12, x: 12

document.writeln(x--);        // x: 12, x become now 11
document.writeln(x);          // x: 11
document.writeln(--x);        // x become now 10, x: 10
</script>
```

## JavaScript Assignment Operators

JavaScript assignment operators assign values to left operand based on right operand. equal (=) operators is used to assign a values.

We have numeric variable:  $x = 10$ ,  $y = 5$  and result.

Operator	Sign	Description	Example	Equivalent to	Results
Assignment	=	Assign value from one operand to another operand value.	result = x	result = x	result = 17
Addition	+=	Addition of operands and finally assign to left operand.	result += x	result = result + y	result = 22
Subtraction	-=	Subtraction of operands and finally assign to left operand.	result -= y	result = result - y	result = 17
Multiplication	*=	Multiplication of operands and finally assign to left operand.	result *= y	result = result * y	result = 85
Division	/=	Division of operands and finally assign to left operand.	result /= y	result = result / y	result = 17
Modulus	%=	Modulus of operands and finally assign to left operand.	result %= y	result = result % y	result = 2
Bitwise AND	&=	AND operator compare two bits values return a results of 1, If both bits are 1. otherwise return 0.	result &= y = 2 & 5 = 0000 0010 & 0000 0101 = 0000 0000 = 0	result = result & y	result = 0
Bitwise OR	=	OR operator compare two bits values and return result of 1, If the bits are complementary. Otherwise return 0.	result  = y = 2   5 = 0000 0010   0000 0101 = 0000 0111 =	result = result   y	result = 7

				7	
Bitwise XOR	$\wedge=$	EXCLUSIVE OR operator compare two bits values and return a results of 1, If any one bits are 1 or either both bits one.	result $\wedge=$ y  = 7 $\wedge$ 5 = 0000 0111 $\wedge$ 0000 0101 = 0000 0010 = 2	result = result $\wedge$ y = 7 $\wedge$ 5 = 0000 0111 $\wedge$ 0000 0101 = 0000 0010 = 2	result = 2
Shift Left	$<<=$	Shift left operator move the bits to a left side.	result $<<=$ y  = 2 $<<=$ 5 = 0000 0010 $<<=$ 0100 0000 = 64	result = result $<<=$ y = 2 $<<=$ 5 = 0000 0010 $<<=$ 0100 0000 = 64	result = 64
Shift Right	$>>=$	Shift left operator move the bits to a left side.	result $>>=$ y  = 2 $>>=$ 5 = 0100 0000 $>>=$ 0000 0010 = 2	result = result $>>=$ y = 2 $>>=$ 5 = 0100 0000 $>>=$ 0000 0010 = 2	result = 2

```

<script>

var x = 17, y = 5;

var result = x; // Assignment to left operand(result) base on right operand(y).
document.writeln(result);

document.writeln(result += x);

document.writeln(result -= y);

document.writeln(result *= y);

document.writeln(result /= y);

document.writeln(result %= y);

document.writeln(result &= y);

```

```

result = 2; // Reassign value
document.writeln(result |= y);
document.writeln(result ^= y);

document.writeln(result <= y);
document.writeln(result >= y);

</script>

```

[Run it... »](#)

## JavaScript Comparison Operators

JavaScript comparison operator determine the two operands satisfied the given condition. Comparison operator return either true or false.

Operator	Sign	Description
Equal	==	If both operands are equal, returns true.
Identical equal	===	If both operands are equal and/or same data type, returns true.
Not equal	!=	If both operands are not equal, returns true.
Identical not equal	!==	If both operands are not equal and/or same data type, returns true.
Greater than	>	If left operand larger than right operand, return true.
Less then	<	If left operand smaller than right operand, return true.
Greater than, equal	>=	If left operand larger or equal than right operand, return true.
Less than, equal	<=	If left operand smaller or equal than right operand, return true.

```

<script>
  document.writeln(5 == 5);    // true
  document.writeln(5 == '5'); // true

```

```

document.writeln(5 === '5'); // false type not same

document.writeln(5 != 10); // true
document.writeln(5 != '10'); // true
document.writeln(5 !== '10'); // true

document.writeln(5 > 10); // false
document.writeln(5 < 10); // true

document.writeln(5 >= 5); // true
document.writeln(5 <= 5); // true

</script>

```

## JavaScript Logical Operators (Boolean Operators)

JavaScript logical operators return boolean result base on operands.

Operator	Sign	Description
Logical AND	&&	If first operand evaluate and return a true, only that evaluate the second operand otherwise skips. Return true if both are must be true, otherwise return false.
Logical OR		Evaluate both operands, Return true if either both or any one operand true, Return false if both are false.
Logical NOT	!	Return the inverse of the given value result true become false, and false become true.

```

<script>

document.writeln((5 == 5) && (10 == 10)); // true
document.writeln(true && false); // false

document.writeln((5 == 5) || (5 == 10)); // true

```

```

document.writeln(true || false);      // true

document.writeln(5 && 10);        // return 10
document.writeln(5 || 10);         // return 5

document.writeln(!5);            // return false
document.writeln(!true);         // return false
document.writeln(!false);        // return true

</script>

```

[Run it... »](#)

## JavaScript Conditional Operator (also call Ternary Operator)

JavaScript conditional operator evaluate the first expression(operand), Base on expression result return either second operand or third operand.

```
answer = expression ? answer1 : answer2;    // condition ? true : false
```

Example

```
document.write((10 == 10) ? "Same value" : "different value");
```

[Run it... »](#)

## JavaScript Bitwise Operators

JavaScript bitwise operators evaluate and perform specific bitwise (32 bits either zero or one) expression.

Operator	Sign	Description
Bitwise AND	&	Return bitwise AND operation for given two operands.
Bitwise OR		Return bitwise OR operation for given two operands.
Bitwise XOR	^	Return bitwise XOR operation for given two operands.
Bitwise NOT	~	Return bitwise NOT operation for given operand.

Bitwise Shift Left	<<	Return left shift of given operands.
Bitwise Shift Right	>>	Return right shift of given operands.
Bitwise Unsigned Shift Right	>>>	Return right shift without consider sign of given operands.

```
<script>

    document.writeln(5 & 10); // return 0, calculation: 0000 0101 & 0000 1010 = 0000 0000
    document.writeln(5 | 10); // return 15, calculation: 0000 0101 | 0000 1010 = 0000 1111
    document.writeln(5 ^ 10); // return 15, calculation: 0000 0101 ^ 0000 1010 = 0000 1111
    document.writeln(~5); // return -6, calculation: ~ 0000 0101 = 1111 1010

    document.writeln(10 << 2); // return 40, calculation: 0000 1010 << 2 = 0010 1000
    document.writeln(10 >> 2); // return 2, calculation: 0000 1010 >> 2 = 0000 0010
    document.writeln(10 >>> 2); // return 2, calculation: 0000 1010 >>> 2 = 0000 0010

</script>
```

[Run it... »](#)

## Miscellaneous Operators

1. typeof
2. delete
3. instanceof
4. new
5. this
6. in

### typeof

JavaScript *typeof* operator return valid data type identifiers as a string of given expression. *typeof* operator return six possible values: "string", "number", "boolean", "object", "function", and "undefined".

typeof expression

typeof(expression)

Example

```
var name = 'Opal Kole';
var age = 48;
var married = true;
var experience = [2010, 2011, 2012, 2013, 2014];
var message = function(){ console.log("Hello world!"); }
var address;

typeof name;      // Returns "string"
typeof age;       // Return "number"
typeof married;   // Return "boolean"
typeof experience; // Return "object"
typeof message;   // Return "function"
typeof address;   // Return "undefined"
```

[Run it... »](#)

## *delete*

JavaScript *delete* operator deletes object property or remove specific element in array.  
If delete is not allow (you can't delete if element not exist, array element undefined etc..) then return false otherwise return true.

```
delete expression;      // delete explicit declare variable

delete object;          // delete object
delete object.property;
delete object[property];

delete array;           // delete array
delete array[index];
```

### Example

```
var address = "63 street Ct.";

delete address;        // Returns false, Using var keyword you can't delete
add = "63 street Ct.';

delete add;           // Returns true, explicit declare you can delete
```

```
var myObj = new Object();
myObj.name = "Opal Kole";
myObj.age = 48;
myObj.married = true;

delete myObj.name;           // delete object property
delete myObj["count"];       // delete object property

var experience = [2010, 2011, 2012, 2013, 2014]; // array elements
delete experience[2];        // delete 2nd index from array elements
console.log(experience);    // [2010, 2011, undefined × 1, 2013, 2014]
```

[Run it... »](#)

## *instanceof*

JavaScript *instanceof* indicate boolean result, Return true, If object is an instance of specific class.

object instanceof class

### Example

```
<script>

var num1 = new Number(15);
document.writeln(num1 instanceof Number);           // Returns true

var num2 = 10;
document.writeln(num2 instanceof Number);           // Return false

document.writeln(true instanceof Boolean);          // false
document.writeln(0 instanceof Number);              // false
document.writeln("") instanceof String);          // false

document.writeln(new Boolean(true) instanceof Boolean); // true
document.writeln(new Number(0) instanceof Number);     // true
document.writeln(new String("") instanceof String);   // true
```

```
</script>
```

[Run it... »](#)

*new*

JavaScript *new* operator to create an instance of the object.

```
var myObj = new Object;  
var myObj = new Object( ); // or you can write  
                        // Object - required, for constructor of the object.
```

```
var arr = new Array( [ argument1, argument2, ..., ..., argumentN ] );  
                        // argument - optional, pass any number of argument in a object.
```

Example

```
var myObj = new Object(); // or you can write: var myObj = new Object;  
myObj.name = "Opal Kole";  
myObj.address = "63 street Ct.";  
myObj.age = 48;  
myObj.married = true;  
  
console.log(myObj);  
// Object {name: "Opal Kole", address: "63 street Ct.", age: 48, married: true}
```

[Run it... »](#)

*this*

JavaScript *this* operator represent current object.

```
this["propertynname"]  
this.propertynname
```

Example

```
function employee(name, address, age, married) {  
    this.name = name;  
    this.address = address;  
    this.age = age;  
    this.married = married;
```

```
}
```

```
var myObj = new employee("Opal Kole", "63 street Ct.", 48, true);
```

```
console.log(myObj);
```

```
// employee {name: "Opal Kole", address: "63 street Ct.", age: 48, married: true}
```

[Run it... »](#)

*in*

JavaScript *in* operator return boolean result if specified property exist in object.

property in object

Example

```
<script>
```

```
var myObj = new Object(); // or you can write: var myObj = new Object;
```

```
myObj.name = "Opal Kole";
```

```
myObj.address = "63 street Ct. ";
```

```
myObj.age = 48;
```

```
myObj.married = true;
```

  

```
document.writeln("name" in myObj); // Returns true
```

```
document.writeln("birthdate" in myObj); // Returns false
```

  

```
                // birthdate property not in myObj
```

```
document.writeln("address" in myObj); // Returns true
```

```
document.writeln("age" in myObj); // Returns true
```

```
document.writeln("married" in myObj); // Returns true
```

```
</script>
```

Like other languages, JavaScript also provides many decision making statements like if, else etc.

## **if statement**

if is used to check for a condition whether its true or not. Condition could be any expression that returns true or false. When condition satisfies then statements following if statement are executed.

Syntax

```
if(condition)
```

```
{  
    statement1  
    statement2  
    ...  
}
```

If you have only one statement to be executed after the if condition then you can drop the curly braces ({ }). For more than one statement use curly braces.

```
if(condition)  
    statement
```

But still its good practice to use curly braces. It will help to easily manage and understand your code.

Example

```
<script>  
  
if(5>4)  
{  
    document.write("yes 5 is greater than 4");  
    document.write("<br />" + "JavaScript is fun");  
    document.write("<br />");  
}  
*****outputs*****  
yes 5 is greater than 4
```

JavaScript is fun

```
******/
```

```
if(true)
```

```
    document.write("Ah! a boolean inside condition. Also no curly braces");
```

```
//see it works without curly braces
```

```
*****outputs*****
```

```
Ah! a boolean inside condition. Also no curly braces
```

```
******/
```

```
if(1==3)
```

```
{
```

```
    document.write("This will not be printed");
```

```
}
```

```
//since condition is false the statements within if will not be executed.
```

```
</script>
```

## else statement

else statements are used with if statements. When if condition gets fail then else statement is executed.

Syntax

```
if(condition)
```

```
{  
    statements  
}  
  
else  
  
{  
    statements  
}
```

You can drop off the curly braces with else too if there is a single statement to be executed within else.

Example

```
<script>  
  
if(3>4)  
{  
    document.write("True");  
}  
  
else  
{  
    document.write("False");  
}  
  
//outputs False
```

```
</script>
```

## else if statement

Suppose there are variety of conditions that you want to check. You can use multiple if statements to do this task. All the if conditions will be checked one by one. But what if you want that if one condition satisfies then don't perform further conditional checks. At this point **else if** statement is what you need.

Syntax

```
if(condition)
{
    statements
}

else if(condition)
{
    statements
}

else
{
    statements
}
```

When every condition fails then final else statement is executed.

Example

```
<script>
```

```
var a=3;  
var b=4;  
  
if(a > b)  
{  
    document.write("a is greater than b");  
}  
  
else if(a < b)  
{  
    document.write("a is smaller than b");  
}  
  
else  
{  
    document.write("Nothing worked");  
}  
  
</script>
```

## switch statement

switch statements do the same task that else if statements do. But use switch statements when conditions are more. In that case, switch statements perform better than else if statements.

### Syntax

```
switch(expression)  
{
```

```
case value-1: statements; break;  
  
case value-2: statements; break;  
  
case value-3: statements; break;  
  
.....  
  
case value-n: statements; break;  
  
default: statements; break;  
}
```

switch evaluates the expression and checks whether it matches with any case. If it matches with any case then statements within that case construct are executed followed by break statement. break statement makes sure that no more case statement gets executed. In case if the expression doesn't match any case value then default case is executed.

There could be any number of statements. No curly braces is needed inside case construct. Also you can omit the break statement in the default construct if default is the last statement within switch.

#### Example

```
<script>
```

```
var a = 7;
```

```
switch(a)

{
    case 1: document.write("one"); break;

    case 2: document.write("two"); break;

    case 3: document.write("three"); break;

    case 4: document.write("four"); break;

    case 5: document.write("five"); break;

    default: document.write("number not found");

}

//outputs three

</script>
```

**Note:** Check that I have dropped the break statement for the default case construct. It will work fine but try to put the default statement above case 1 and see the result. Also try this program removing all the break statements.

**case** value can be any number, character or string. Keep this in mind.

Example

```
<script>
```

```
var i = 'h';

switch(i)
```

```
{
```

```
case 'a': document.write("a found"); break;  
case 'b': document.write("b found"); break;  
case 'c': document.write("c found"); break;  
case 'h': document.write("h found"); break;  
case 'string': document.write("string found"); break;  
default: document.write("nothing found");  
}  
  
//outputs h found  
</script>
```

## Ternary Operator (?:)

Ternary operator is preferred by many programmers. It contains 3 operands that's why the name ternary operator.

### Syntax

```
condition ? if-true-execute-this : if-false-execute-this;
```

If condition goes right then statement before colon is executed and if false then statement after colon is executed. Multiple statements are not allowed. But you can execute multiple statements by making a function and calling the function if the condition goes true or false accordingly.

### Example

```
<script>  
  
true ? document.write("True value found") : document.write("False value found");  
  
document.write("<br />");  
  
//outputs True value found
```

```
(5>4 && 4==3) ? document.write("True") : document.write("False");

document.write("<br />");

//outputs False

var a = (true) ? 1 : 2;

document.write(a);

//outputs 1

</script>
```

## Nested if else or switch statements

You can insert if or else or switch condition within another if or else or switch condition. This is called nesting. You can nest if, else or switch conditions up to any number of level but doing so will make you code more confusing. So use it wisely.

### Syntax

```
if(condition)

{
    if(other condition)

    {
        statements;
    }
}
```

```
else
{
    if(other condition)
    {
        statements;
    }
    else
    {
        switch(expression)
        {
            case value-1: statements; break;
            .....
        }
    }
}
```

## Type Casting in JavaScript

Converting a data type into another is known as type casting. Sometimes there is a need to convert the data type of one value to another.

### **typeof**

typeof operator is used to return the data type of an operand.

#### Syntax

typeof operand;

Example

```
<script>
```

```
var a;
```

```
document.write(typeof a); //outputs undefined
```

```
document.write("<br />");
```

```
document.write(typeof "SyntaxPage"); //outputs string
```

```
document.write("<br />");
```

```
document.write(typeof 7); //outputs number
```

```
document.write("<br />");
```

```
document.write(typeof true); //outputs boolean
```

```
document.write("<br />");
```

```
document.write(typeof document); //outputs object
```

```
document.write("<br />");
```

```
document.write(typeof document.write); //outputs function
```

```
document.write("<br />");

</script>
```

The code is self explanatory. The variable a has not been initialized that's why its type is showing as undefined. Try to assign some value to it and see the result.

## Converting to Boolean

To convert a value to boolean data type, just pass the value to Boolean function.

Syntax

```
Boolean(value);
```

Example

```
<script>

var b = 1;

b = Boolean(b);

document.write(b + " : " + typeof b); //outputs true : boolean

</script>
```

## Converting to String

To convert a value to string data type, just pass the value to String function.

Syntax

String(value);

Example

```
<script>

var s = 1;

s= String(s);

document.write(s + " : " + typeof s); //outputs 1 : string

</script>
```

## Converting to Number

Many times string needs to get converted into numbers. To convert a value to number data type, just pass the value to Number function.

Syntax

Number(value);

Example

```
<script>

var n1 = true;

var n2 = "1str";
```

```
var n3 = "123";  
  
n1 = Number(n1);  
  
n2 = Number(n2);  
  
n3 = Number(n3);  
  
  
  
document.write(n1 + " : " + typeof n1); //outputs 1 : number  
  
document.write("<br />");  
  
  
  
document.write(n2 + " : " + typeof n2); //outputs NaN : number  
  
document.write("<br />");  
  
  
  
document.write(n3 + " : " + typeof n3); //outputs 123 : number  
  
document.write("<br />");  
  
  
  
 </script>
```

true is represented as 1 in numeric so n1 after conversion stores value 1.  
"1str" after conversion doesn't represent a valid number that's why the output is NaN. NaN means not a number. Its a language construct.  
"123" after conversion represents valid number 123.

There are other methods to convert string to number format.

## **parseInt**

parseInt() function is used to convert strings values into numbers. It works differently than Number() function. It doesn't work for boolean or other data-types values. For others values or

string that doesn't contain numbers in it, parseInt() will just output NaN. Examples will clear this idea.

### Syntax

```
parseInt(string);
```

### Example

```
<script>
```

```
var n1 = true;
```

```
var n2 = "1str";
```

```
var n3 = "123";
```

```
var n4 = "str123str1";
```

```
n1 = parseInt(n1);
```

```
n2 = parseInt(n2);
```

```
n3 = parseInt(n3);
```

```
n4 = parseInt(n4);
```

```
document.write(n1 + " : " + typeof n1); //outputs NaN : number
```

```
document.write("<br />");
```

```
document.write(n2 + " : " + typeof n2); //outputs 1: number
```

```
document.write("<br />");
```

```
document.write(n3 + " : " + typeof n3); //outputs 123 : number  
document.write("<br />");  
  
document.write(n3 + " : " + typeof n4); //outputs 123 : number  
document.write("<br />");  
  
</script>
```

## parseFloat

parseFloat() function is used to convert strings values into floating point numbers. It works similar to parseInt() function with an exception of handling decimal point numbers. For example, 1.23 is represented as 123e-2. parseInt("123e-2") will result in 123 but parseFloat("123e-2") will give 1.23.

### Syntax

```
parseFloat(string);
```

### Example

```
<script>  
  
var n1 = "123e-2";  
  
var n2 = "1str";  
  
var n3 = "123";  
  
var n4 = "str123str1";
```

```
n1 = parseFloat(n1);

n2 = parseFloat(n2);

n3 = parseFloat(n3);

n4 = parseFloat(n4);

document.write(n1 + " : " + typeof n1); //outputs 1.23 : number
document.write("<br />");

document.write(n2 + " : " + typeof n2); //outputs 1: number
document.write("<br />");

document.write(n3 + " : " + typeof n3); //outputs 123 : number
document.write("<br />");

document.write(n3 + " : " + typeof n4); //outputs 123 : number
document.write("<br />");

</script>
```

## Loops in JavaScript

Loops are used to execute a specific statement for a given number of times. Loops are always followed by some condition. JavaScript provides all the basic loops that other programming languages have.

## while loop

while loop checks for a given condition to be true to execute statements that it contains within it. When the condition becomes false, while loop break the execution of statements.

### Syntax

```
initializer;  
  
while(condition)  
  
{  
  
    statement-1;  
  
    statement-2;  
  
    .....  
  
    modifier;  
  
}
```

initializer is a variable that is tested within the condition. modifier modifies the initializer so that the condition becomes false at some point.

**Note:** Make sure that your condition goes false somehow else you will be in an infinite loop.

### Example

```
<script>  
  
var i = 0;  
  
while(i<6)  
  
{  
  
    document.write(i);  
  
    document.write("<br />");  
  
    i++;
```

```
}
```

```
</script>
```

## do while loop

do while loop first execute the statements that it contain and then check for a condition. So even if the condition is false do-while loop is going to run at least once.

### Syntax

```
initializer;  
  
do  
  
{  
  
    statement-1;  
  
    statement-2;  
  
    .....  
  
    modifier;  
  
}  
  
while(condition);
```

### Example

```
<script>
```

```
var i = 5;
```

```
do
```

```
{  
    document.write(i);  
  
    document.write("<br />");  
  
    i++;  
  
}  
  
while(i<6);  
  
</script>
```

## for loop

for loop is generally used when the number of iterations/repetitions are already known. Its not that such tasks can't be done by while loop but its just a matter of preference.

### Syntax

```
for(initializer; condition; modifier)  
{  
    statement-1;  
  
    ....  
  
}
```

Here, initializer is a variable that is defined once in the beginning of a for loop. It is optional and can be ignored. condition is checked after initialization. Then statements are executed and finally modifier modifies the initializer. After that again condition is checked and the process continues. Modifier can also be ignored.

**Note:** The only required thing within for loop is condition.

You can drop the curly braces if only one statement is within the for loop.

### Syntax

```
<script>

for(var i=0;i<10;i++)

{

    document.write(i);

    document.write("<br />");

}

</script>
```

## Nested Loops

Any loop can be inserted inside any other loop. This is called nesting. for loop can be nested inside while or for loop. Similarly while loop can be nested inside for loop or another while loop.

### Syntax

```
for(initializer; condition; modifier)

{

    while(condition)

    {

        statements;

    }

}
```

### Example

```
<script>
```

```
var i,j;  
  
for(i=0;i<3;i++)  
{  
    document.write("Outer Loop : " + i);  
  
    document.write("<br />");  
  
    for(j=0;j<3;j++)  
    {  
        document.write("Inner Loop : " + j);  
  
        document.write("<br />");  
    }  
  
    document.write("<br />");  
}  
  
*****output*****
```

Outer Loop : 0

Inner Loop : 0

Inner Loop : 1

Inner Loop : 2

Outer Loop : 1

Inner Loop : 0

Inner Loop : 1

Inner Loop : 2

Outer Loop : 2

Inner Loop : 0

Inner Loop : 1

Inner Loop : 2

\*\*\*\*\*\*/

</script>

Can you guess how above loop works? First outer for loop initializes i, checks for the condition then executes its first two statements. After that inner for loop starts to work. After executing its statements 3 times, outer for loop is given the control again. And the process is repeated.

## Loops Control Statements in JavaScript

Loop control statements deviates a loop from its normal flow. All the loop control statements are attached to some condition inside the loop.

### **break statement**

break statement is used to break the loop on some condition.

Syntax

```
for(initializer, condition, modifier)
```

```
{
```

```
    if(condition)
```

```
{  
    break;  
}  
  
statements;  
  
}  
  
  
//same thing is valid for while loop  
  
while(condition)  
  
{  
    if(condition)  
    {  
        break;  
    }  
  
    statements;  
}
```

Example

```
<script>  
  
for(var i=1;i<10;i++)  
  
{  
    if(i==7) break;
```

```
document.write(i + "<br />");

}

*****output*****
1
2
3
4
5
6
*****/
</script>
```

for loop should print from 1 - 10. But inside loop we have given a condition that if value of i equals to 7, then break out of loop. I have left the curly braces within the if condition because there is only one statement to be executed.

**Remember break burst the nearest for loop within which it is contained.** To understand what it means check out the next example.

Example

```
<script>

for(var i=1;i<=3;i++)
{
    document.write("Outer Loop : " + i + "<br />");
```

```
for(var j=1;j<=3;j++)  
{  
    document.write("Inner Loop : " + j + "<br />");  
    if(j==2)  
    {  
        break;  
    }  
}  
  
document.write("<br />");  
}  
  
*****output*****
```

Outer Loop : 1

Inner Loop : 1

Inner Loop : 2

Outer Loop : 2

Inner Loop : 1

Inner Loop : 2

Outer Loop : 3

Inner Loop : 1

Inner Loop : 2

```
******/
```

```
</script>
```

## continue statement

continue statement is used to skip an iteration of a loop.

Syntax

```
for(initializer, condition, modifier)
```

```
{
```

```
    if(condition)
```

```
{
```

```
        continue;
```

```
}
```

```
    statements;
```

```
}
```

```
//same thing is valid for while loop
```

```
while(condition)
```

```
{
```

```
    if(condition)
```

```
{  
    continue;  
}  
  
statements;  
}
```

### Example

```
<script>  
  
for(var i=0;i<10;i++)  
{  
    if(i==3 || i==7) continue;  
  
    document.write(i + "<br />");  
  
}  
  
*****output*****  
  
0  
1  
2  
4  
5  
6
```

```
8  
9  
******/  
</script>
```

In the code above when the value of i becomes 3 or 7, continue forces the loop to skip without further executing any instructions inside the for loop in that current iteration.

**Remember continue skips the nearest loop iteration within which it is contained.** To understand what it means check out the next example.

Example

```
<script>  
  
for(var i=1;i<=3;i++)  
{  
    document.write("Outer Loop : " + i + "<br />");  
  
    for(var j=1;j<=3;j++)  
    {  
        if(j==2)  
        {  
            continue;  
        }  
        document.write("Inner Loop : " + j + "<br />");  
    }  
}
```

```
}

document.write("<br />");

}

*****output*****
```

Outer Loop : 1

Inner Loop : 1

Inner Loop : 3

Outer Loop : 2

Inner Loop : 1

Inner Loop : 3

Outer Loop : 3

Inner Loop : 1

Inner Loop : 3

```
******/
```

</script>

When value of j inside inner loop becomes 2, it skips that iteration.

## Mathematical Constants in JavaScript

**Math** object in JavaScript provides various mathematical constants and mathematical functions. Below is the list of various constants.

Constant	Description	Value
Math.PI	Constant PI	3.141592653589793
Math.E	Euler's Constant	2.718281828459045
Math.LN2	Natural log of 2	0.6931471805599453
Math.LN10	Natural log of 10	2.302585092994046
Math.LOG2E	Base 2 log of E	1.4426950408889634
Math.LOG10E	Base 10 log of E	0.4342944819032518
Math.SQRT1_2	Square root of 0.5	0.7071067811865476
Math.SQRT2	Square root of 2	1.4142135623730951

Example

```
<script>

var x = Math.PI;

document.write(x); //outputs 3.141592653589793

</script>
```

## Commonly used Mathematical Methods in JavaScript

Some commonly used methods that are used mostly in JavaScript are as follows:

Method	Description
Math.abs(number)	returns an absolute value
Math.sqrt(number)	returns square root of a number
Math.pow(a, b)	returns power value a raised to the power b
Math.log(number)	returns natural log value
Math.exp(number)	returns Euler's constant raised to power of specified number
Math.max(a, b)	returns larger of two numbers
Math.min(a, b)	returns smaller of two numbers

Example

```
<script>

var a = Math.abs(-7); //7

document.write(a);

document.write("<br />");

a = Math.sqrt(64); //8

document.write(a);

document.write("<br />");
```

```
a = Math.pow(2, 3); //2x2x2 = 8
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.log(20); //2.995732273553991
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.exp(5); //148.41315910257657
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.max(5,10); //10
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.min(5,10); //5
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.min(-50,-49); //-50
```

```
document.write(a);
```

```
</script>
```

## Trigonometric Methods in JavaScript

Trigonometric Methods are rarely used but still sometimes helpful. Some of the trigonometric methods used are given below:

Method	Description
Math.sin(number)	returns sine value
Math.cos(number)	returns cosine value
Math.tan(number)	returns tangent value
Math.asin(number)	returns arc sine value
Math.acos(number)	returns arc cosine value
Math.atan(number)	returns arc tangent value

Example

```
<script>

var a = Math.sin(0); //sin 0 degree = 0

document.write(a);

document.write("<br />");
```

```
a = Math.cos(0); //cos 0 degree = 1
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.tan(0); //tan 0 degree = 0
```

```
document.write(a);
```

```
</script>
```

## Rounding Numbers in JavaScript

You can round off numbers using the functions given in the table.

Method	Description
Math.ceil(number)	returns rounded up integer value
Math.floor(number)	returns rounded down integer value
Math.round(number)	returns nearest integer value

Example

```
<script>
```

```
var a = Math.ceil(12.4); //13
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.floor(12.4); //12
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.round(12.4); //12
```

```
document.write(a);
```

```
document.write("<br />");
```

```
a = Math.round(12.5); //13
```

```
document.write(a);
```

```
</script>
```

## Rounding Number up to specific level of precision in JavaScript

While rounding a number using round() method, you will always end up getting some integer. Not all the time you want this. Sometimes you need to get a value up to some specific decimal level or precision. For example, Math.PI returns 3.141592653589793. What if you want to get this value up to 2 decimal places (3.14)? For doing so, you can multiply the floating point number with 100. Then use the round() method and then divide the number again by 100. Below is a formula that you can use.

### Syntax

```
(Math.round(number * precision factor)) / precision factor
```

Precision factor represents the decimal level and is equal to 10 raise to the power of the decimal places. For example if you want 2 decimal places, precision factor will be 100( $10^2$ ), for 3 decimal places it should be 1000( $10^3$ ).

### Example

```
<script>

var p = Math.round(Math.PI); //3

document.write(p);

document.write("<br />");

//for 2 decimal places

var p = Math.PI * 100;

p = Math.round(p);

p /= 100;

document.write(p);

document.write("<br />");

//for 4 decimal places

var p = (Math.round(Math.PI * 10000))/10000;

document.write(p);

</script>
```

## Random Number in JavaScript

**Math.random()** method is used to generate a random floating point number between 0.0 and 1.0. You can always increase its range by multiplying it to a number. For example, multiplying it

with 10 will increase its range from 0.0 to 10.0. And to get an integer range, use Math.ceil() so that the range actually becomes 1 to 10. You can use Math.floor() to make the range between 0 and 9.

#### Syntax

```
Math.random();
```

#### Example

```
<script>

var r = Math.random();

r *= 10;

r = Math.ceil(r); //returns any number between 1-10

document.write(r);

document.write("<br />");

r = Math.floor(Math.random()*10); //returns any number between 0-9

document.write(r);

</script>
```

## Infinity in JavaScript

Infinity is a global property (variable) that is used to represent infinite values in JavaScript.

#### Example

```
<script>
```

```
var i = 100 / 0;  
  
document.write(i); //Infinity  
  
</script>
```

There are also two other global properties in JavaScript, POSITIVE\_INFINITY and NEGATIVE\_INFINITY but they are rarely used.

## NaN in JavaScript

NaN is a global property which means "Not a Number".

Example

```
<script>  
  
var n = "string" * 10; //NaN  
  
document.write(n);  
  
</script>
```

## Working with Strings in JavaScript

String plays an important role in every programming languages. JavaScript is no exception. Also every character in JavaScript is represented by some number. That numerical unicode is known as charcode of that character. Example, 'A' is represented by 65.

Here are few functions that can be useful to help you out with your day to day coding work.

### String Concatenation

String concatenation refers to combining one string to another. Simplest way to join strings is using '+' (plus) operator. But there is also a method **concat()** which can be used to unite strings together.

### Syntax

```
//using concat method  
  
string1.concat(string2, string3, ...);  
  
  
//using + operator  
  
string1 + string2 + string3
```

### Example

```
<script>  
  
  
  
var str = "Syntax";  
  
str = str.concat("Page"); //SyntaxPage  
  
document.write(str);  
  
document.write("<br />");  
  
  
  
str = "JavaScript";  
  
str = str.concat(" is", " fun", "<br />");  
  
document.write(str); //JavaScript is fun  
  
  
  
str = "JavaScript " + "is " + "awesome" + "<br />";
```

```
document.write(str); //JavaScript is awesome
```

```
</script>
```

## Splitting Strings

Various methods are available in JavaScript to get whole or some part of strings. Few commonly used methods are as follows:

### **substring() method**

substring method is used to get arbitrary string values between the index positions passed to it as parameters.

#### Syntax

```
string.substring(start,[end])
```

First parameter specifies from where to copy the string. Second optional parameter specifies till where to copy. Also, character specified by second parameter is not included. If second parameter is not given then whole string after start position will be copied.

#### Example

```
<script>

var str = 'SyntaxPage';

var sub = str.substring(6); //Page

document.write(sub);

document.write("<br />")

str = 'JavaScript';
```

```
sub = str.substring(4,7); //Src  
  
document.write(sub);  
  
//element at 4th position is excluded  
  
</script>
```

### **substr() method**

substr method is somewhat similar to substring() method with just one difference. The second parameter in the substr() method specifies the length of the string to be copied, not the end position.

#### Syntax

```
string.substr(start,length)
```

#### Example

```
<script>  
  
var str = 'SyntaxPage';  
  
var sub = str.substr(6); //Page  
  
document.write(sub);  
  
document.write("<br />")  
  
  
str = 'JavaScript';  
  
sub = str.substr(4,3); //Scr
```

```
document.write(sub);
```

```
</script>
```

## **slice() method**

slice method is used to get characters from a string between specified indexes passed to it as parameters.

### Syntax

```
string.slice(start, [end])
```

The element at the 'end' index is not included. If end index is not specified then the result will be from start index till the end of the string.

### Example

```
<script>

var str = "12345";

var sub = str.slice(1, 4); //234

document.write(sub);

document.write("<br />");

sub = str.slice(1); //2345

document.write(sub);
```

```
</script>
```

## split() method

split() method breaks a string into an array of string. It splits the string from the separator string boundaries that is passed to it as parameter.

### Syntax

```
string.split(separator,[size])
```

If no separator string is given then it will break the string into an array having just one element which is string itself.

If optional size parameter is given then array formed will be of given size parameter ignoring the rest of the characters.

### Example

```
<script>

var str = "JavaScript is fun";

var sub = str.split(" ");

document.write(sub); //JavaScript,is,fun

document.write("<br />");

sub = str.split(" ", 1);

document.write(sub); //JavaScript

document.write("<br />");
```

```
sub = str.split();  
  
document.write(sub[0]); //JavaScript is fun  
  
</script>
```

## Changing Case

### **toLowerCase() method**

toLowerCase() method is used to change the case of the string to lower case.

Syntax

```
string.toLowerCase()
```

Example

```
<script>  
  
var str = "JavaScript is Awesome";  
  
var lcase = str.toLowerCase();  
  
document.write(lcase);  
  
</script>
```

### **toUpperCase() method**

toUpperCase() method is used to change the case of the string to upper case.

Syntax

```
string.toUpperCase()
```

Example

```
<script>

var str = "JavaScript is Awesome";

var ucase = str.toUpperCase();

document.write(ucase);

</script>
```

## Finding Characters

### **indexOf() method**

indexOf() method is used to search for a given substring in the parent string and returns substring position.

Syntax

```
string.indexOf(needle,[fromIndex])
```

Optional second parameter decides from where to begin searching the string. It starts searching from the beginning if optional second parameter is not given. In case needle(substring) is not found then it returns -1.

Example

```
<script>
```

```
var str = "JavaScript is an Awesome Script";
```

```
var pos = str.indexOf("Script"); //4  
  
document.write(pos);  
  
document.write("<br />");  
  
  
  
var pos = str.indexOf("Script",5); //25  
  
document.write(pos);  
  
  
  
</script>
```

### **lastIndexOf() method**

lastIndexOf() method is used to search for a given substring beginning the search from the end. It returns needle's position. In case needle(substring) is not found then it returns -1.

#### Syntax

```
string.lastIndexOf(needle,[fromIndex])
```

It starts searching from the end if optional second parameter is not given. In case needle(substring) is not found then it returns -1.

#### Example

```
<script>  
  
  
  
var str = "JavaScript is an Awesome Script";  
  
var pos = str.lastIndexOf("Script"); //25  
  
document.write(pos);
```

```
var pos = str.lastIndexOf("Script",24); //4  
  
document.write(pos);  
  
</script>
```

### **search() method**

search() returns the index position of the needle if found else it returns -1. It looks somewhat similar to indexOf() method but search() method works with regular expressions. Regular expressions will be discussed later.

#### Syntax

```
string.search(regexp)
```

#### Example

```
<script>  
  
var str = "JavaScript is an Awesome Script";  
  
var pos = str.search("Script"); //4  
  
document.write(pos);  
  
</script>
```

### **match() method**

match() returns the substring(needle) if found in the main string else it returns null. match() method also works with regular expressions. Performance wise indexOf() method is faster than match() and search().

#### Syntax

```
string.match(regexp)
```

#### Example

```
<script>

var str = "JavaScript is an Awesome Script";

var pos = str.match("Script"); //Script

document.write(pos);

</script>
```

#### charAt() method

charAt() returns the character at a particular index which is passed to it as an argument.

```
string.charAt(index)
```

#### Example

```
<script>

var str = "JavaScript is an Awesome Script";

var c = str.charAt(4); //S
```

```
document.write(c);
```

```
</script>
```

### **charCodeAt() method**

charCodeAt() returns the unicode numeric value of a character at a particular index which is passed to it as an argument.

```
string.charCodeAt(index)
```

#### Example

```
<script>

var str = "JavaScript is an Awesome Script";

var u = str.charCodeAt(4); //83

document.write(u);

</script>
```

### **Replacing word in a string**

#### **replace() method**

replace() method is used to find and replace some or all occurrence of a string/character with some new specified string/character within a string. To replace more than one value within a string pass regular expression as first argument.

#### Syntax

```
string.replace(regular_expression/string, 'new_string')
```

Example

```
<script>

var str = "JavaScript is Awesome";

var newStr = str.replace("Awesome", "Amazing");

document.write(newStr);

</script>
```

## Summary of String Methods in JavaScript

Method	Description
str.concat(str1, str2,...)	combines one or more string
str.substring(start,[end])	returns substring from start to end unless end index position is specified
str.substr(start,len)	returns substring from start to end unless number of characters(length) is specified
str.slice(start, [end])	returns substring between specified indexes
str.split(sep,[size])	breaks a string into an array from the separator string boundaries
str.toLowerCase()	returns lowercase string

str.toUpperCase()	returns uppercase string
str.indexOf(needle,[fromIndex])	returns character present at specified index beginning the search from start. Second parameter decides from where to begin search
str.lastIndexOf(needle,[fromIndex])	returns character present at specified index beginning the search from end. Second parameter decides from where to begin search
str.search(regexp)	returns the index position of the substring if found else it returns -1
str.match(regexp)	returns the substring if found else it returns null.
str.charAt(index)	returns the character at a particular index which is passed to it as an argument
str.charCodeAt(index)	returns the unicode numeric value of a character at a particular index which is passed to it as an argument
str.replace(regular_expression/string, "new_string")	returns string after replacing some or all occurrence of old string with new string within a given string.

## Date and Time in JavaScript

Date object in JavaScript can be used to get date and time. Date object has various methods to exploit date and time. To get date and time use new keyword followed by with Date object constructor.

### Syntax

```
//get date and time
```

```
new Date();
```

```
//set date and time  
new Date(year, month, date, hours, minutes, seconds, milliseconds);
```

Example

```
<script>  
  
var now = new Date();  
  
document.write(now);
```

//Fri Feb 20 2015 21:12:09 GMT+0530 (India Standard Time)

```
document.write('<br />');  
  
now = new Date(2005, 11, 21, 12, 30, 30, 0);  
  
document.write(now);
```

//Wed Dec 21 2005 12:30:30 GMT+0530 (India Standard Time)

```
</script>
```

## Working with Date

To get and set date, Date objects has various methods.

date.getDay()	returns the day number (0-6, 0 for sun, 1 for mon and so on)
---------------	--------------------------------------------------------------

date.getDate()	returns the date (1 to 31)
date.getMonth()	returns the month (0-11, 0 for jan and so on)
date.getFullYear()	returns 4 digit year
date.setDate(value)	sets the date (1 to 31)
date.setMonth(value)	sets the month (0-11, 0 for jan and so on)
date.setFullYear()	sets 4 digit year

Example

```
<script>

var days = ['Sun', 'Mon', 'Tues', 'Wed', 'Thu', 'Fri', 'Sat'];

var months = ['Jan', 'Feb', 'March', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];

var d = new Date();

var day = d.getDay();

day = days[day];

var date = d.getDate();
```

```
var month = d.getMonth();  
month = months[month];  
  
var year = d.getFullYear();  
  
document.write(day + " " + date + " " + month + " " + year);  
  
</script>
```

## Working with Time

To get and set time, Date objects has various methods.

date.getHours()	returns hour in 24-hour format (0-23)
date.getMinutes()	returns minute (0-59)
date.getSeconds()	returns second (0-59)
date.getMilliseconds()	returns millisecond (0-999)
date.setHours()	sets hour in 24-hour format (0-23)
date.setMinutes()	sets minute (0-59)
date.setSeconds()	sets second (0-59)
date.setMilliseconds()	sets millisecond (0-999)

Example

```
<script>

var t = new Date();

var hour = t.getHours();

var min = t.getMinutes();

var sec = t.getSeconds();

var milli = t.getMilliseconds();

document.write(hour + ":" + min + ":" + sec + ":" + milli);

</script>
```

## Working with Strings in JavaScript

String plays an important role in every programming languages. JavaScript is no exception. Also every character in JavaScript is represented by some number. That numerical unicode is known as charcode of that character. Example, 'A' is represented by 65.

Here are few functions that can be useful to help you out with your day to day coding work.

### String Concatenation

String concatenation refers to combining one string to another. Simplest way to join strings is using '+' (plus) operator. But there is also a method **concat()** which can be used to unite strings together.

### Syntax

```
//using concat method  
  
string1.concat(string2, string3, ...);  
  
  
//using + operator  
  
string1 + string2 + string3
```

### Example

```
<script>  
  
  
  
var str = "Syntax";  
  
str = str.concat("Page"); //SyntaxPage  
  
document.write(str);  
  
document.write("<br />");  
  
  
  
str = "JavaScript";  
  
str = str.concat(" is", " fun", "<br />");  
  
document.write(str); //JavaScript is fun  
  
  
  
str = "JavaScript " + "is " + "awesome" + "<br />";
```

```
document.write(str); //JavaScript is awesome
```

```
</script>
```

## Splitting Strings

Various methods are available in JavaScript to get whole or some part of strings. Few commonly used methods are as follows:

### **substring() method**

substring method is used to get arbitrary string values between the index positions passed to it as parameters.

#### Syntax

```
string.substring(start,[end])
```

First parameter specifies from where to copy the string. Second optional parameter specifies till where to copy. Also, character specified by second parameter is not included. If second parameter is not given then whole string after start position will be copied.

#### Example

```
<script>

var str = 'SyntaxPage';

var sub = str.substring(6); //Page

document.write(sub);

document.write("<br />")

str = 'JavaScript';
```

```
sub = str.substring(4,7); //Src  
  
document.write(sub);  
  
//element at 4th position is excluded  
  
</script>
```

### **substr() method**

substr method is somewhat similar to substring() method with just one difference. The second parameter in the substr() method specifies the length of the string to be copied, not the end position.

#### Syntax

```
string.substr(start,length)
```

#### Example

```
<script>  
  
var str = 'SyntaxPage';  
  
var sub = str.substr(6); //Page  
  
document.write(sub);  
  
document.write("<br />")  
  
  
str = 'JavaScript';  
  
sub = str.substr(4,3); //Scr
```

```
document.write(sub);
```

```
</script>
```

## **slice() method**

slice method is used to get characters from a string between specified indexes passed to it as parameters.

### Syntax

```
string.slice(start, [end])
```

The element at the 'end' index is not included. If end index is not specified then the result will be from start index till the end of the string.

### Example

```
<script>

var str = "12345";

var sub = str.slice(1, 4); //234

document.write(sub);

document.write("<br />");

sub = str.slice(1); //2345

document.write(sub);
```

```
</script>
```

## split() method

split() method breaks a string into an array of string. It splits the string from the separator string boundaries that is passed to it as parameter.

### Syntax

```
string.split(separator,[size])
```

If no separator string is given then it will break the string into an array having just one element which is string itself.

If optional size parameter is given then array formed will be of given size parameter ignoring the rest of the characters.

### Example

```
<script>

var str = "JavaScript is fun";

var sub = str.split(" ");

document.write(sub); //JavaScript,is,fun

document.write("<br />");

sub = str.split(" ", 1);

document.write(sub); //JavaScript

document.write("<br />");
```

```
sub = str.split();  
  
document.write(sub[0]); //JavaScript is fun  
  
</script>
```

## Changing Case

### **toLowerCase() method**

toLowerCase() method is used to change the case of the string to lower case.

Syntax

```
string.toLowerCase()
```

Example

```
<script>  
  
var str = "JavaScript is Awesome";  
  
var lcase = str.toLowerCase();  
  
document.write(lcase);  
  
</script>
```

### **toUpperCase() method**

toUpperCase() method is used to change the case of the string to upper case.

Syntax

string.toUpperCase()

Example

```
<script>

var str = "JavaScript is Awesome";

var ucase = str.toUpperCase();

document.write(ucase);

</script>
```

## Finding Characters

### indexOf() method

indexOf() method is used to search for a given substring in the parent string and returns substring position.

Syntax

```
string.indexOf(needle,[fromIndex])
```

Optional second parameter decides from where to begin searching the string. It starts searching from the beginning if optional second parameter is not given. In case needle(substring) is not found then it returns -1.

Example

```
<script>
```

```
var str = "JavaScript is an Awesome Script";
```

```
var pos = str.indexOf("Script"); //4  
  
document.write(pos);  
  
document.write("<br />");  
  
  
  
var pos = str.indexOf("Script",5); //25  
  
document.write(pos);  
  
  
  
</script>
```

### **lastIndexOf() method**

lastIndexOf() method is used to search for a given substring beginning the search from the end. It returns needle's position. In case needle(substring) is not found then it returns -1.

#### Syntax

```
string.lastIndexOf(needle,[fromIndex])
```

It starts searching from the end if optional second parameter is not given. In case needle(substring) is not found then it returns -1.

#### Example

```
<script>  
  
  
  
var str = "JavaScript is an Awesome Script";  
  
var pos = str.lastIndexOf("Script"); //25  
  
document.write(pos);
```

```
var pos = str.lastIndexOf("Script",24); //4  
  
document.write(pos);  
  
</script>
```

### **search() method**

search() returns the index position of the needle if found else it returns -1. It looks somewhat similar to indexOf() method but search() method works with regular expressions. Regular expressions will be discussed later.

#### Syntax

```
string.search(regexp)
```

#### Example

```
<script>  
  
var str = "JavaScript is an Awesome Script";  
  
var pos = str.search("Script"); //4  
  
document.write(pos);  
  
</script>
```

### **match() method**

match() returns the substring(needle) if found in the main string else it returns null. match() method also works with regular expressions. Performance wise indexOf() method is faster than match() and search().

#### Syntax

```
string.match(regexp)
```

#### Example

```
<script>

var str = "JavaScript is an Awesome Script";

var pos = str.match("Script"); //Script

document.write(pos);

</script>
```

#### charAt() method

charAt() returns the character at a particular index which is passed to it as an argument.

```
string.charAt(index)
```

#### Example

```
<script>

var str = "JavaScript is an Awesome Script";

var c = str.charAt(4); //S
```

```
document.write(c);
```

```
</script>
```

### **charCodeAt() method**

charCodeAt() returns the unicode numeric value of a character at a particular index which is passed to it as an argument.

```
string.charCodeAt(index)
```

#### Example

```
<script>

var str = "JavaScript is an Awesome Script";

var u = str.charCodeAt(4); //83

document.write(u);

</script>
```

### **Replacing word in a string**

#### **replace() method**

replace() method is used to find and replace some or all occurrence of a string/character with some new specified string/character within a string. To replace more than one value within a string pass regular expression as first argument.

#### Syntax

```
string.replace(regular_expression/string, 'new_string')
```

Example

```
<script>

var str = "JavaScript is Awesome";

var newStr = str.replace("Awesome", "Amazing");

document.write(newStr);

</script>
```

## Summary of String Methods in JavaScript

Method	Description
str.concat(str1, str2,...)	combines one or more string
str.substring(start,[end])	returns substring from start to end unless end index position is specified
str.substr(start,len)	returns substring from start to end unless number of characters(length) is specified
str.slice(start, [end])	returns substring between specified indexes
str.split(sep,[size])	breaks a string into an array from the separator string boundaries
str.toLowerCase()	returns lowercase string

str.toUpperCase()	returns uppercase string
str.indexOf(needle,[fromIndex])	returns character present at specified index beginning the search from start. Second parameter decides from where to begin search
str.lastIndexOf(needle,[fromIndex])	returns character present at specified index beginning the search from end. Second parameter decides from where to begin search
str.search(regexp)	returns the index position of the substring if found else it returns -1
str.match(regexp)	returns the substring if found else it returns null.
str.charAt(index)	returns the character at a particular index which is passed to it as an argument
str.charCodeAt(index)	returns the unicode numeric value of a character at a particular index which is passed to it as an argument
str.replace(regular_expression/string, "new_string")	returns string after replacing some or all occurrence of old string with new string within a given string.

## Different Types of Conditional Statements

There are mainly three types of conditional statements in JavaScript.

1. If statement
2. If...Else statement
3. If...Else If...Else statement

## If statement

Syntax:

```
if (condition)
```

```
{  
lines of code to be executed if condition is true  
}
```

You can use If statement if you want to check only a specific condition.

Try this yourself:

This code is editable. Click Run to Execute



```
<html>  
<head>  
    <title>IF Statements!!!</title>  
    <script type="text/javascript">  
        var age = prompt("Please enter your age");  
        if(age>=18)  
            document.write("You are an adult <br />");  
        if(age<18)  
            document.write("You are NOT an adult <br />");  
    </script>  
</head>  
<body>  
</body>
```

</html>