# Loops in JavaScript

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. For example, suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

**Iterative Method**

Iterative method to do this is to write the document.write() statement 10 times.

## 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 | Squre 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 contsant 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

Trignometric 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 seach() 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.charAt(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 seach() 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.charAt(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 Statments!!!</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>
```