(the source if this document: w3schools.com)

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.
Given that **y=5**, the table below explains the arithmetic operators:

Operator	Description	Example	Result of x	Result of y
+	Addition	x=y+2	7	5
-	Subtraction	x=y-2	3	5
*	Multiplication	x=y*2	10	5
/	Division	x=y/2	2.5	5
%	Modulus (division remainder)	x=y%2	1	5
++	Increment	x=++y	6	6
		x=y++	5	6
--	Decrement	x=--y	4	4
		x=y--	5	4

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.
Given that **x=10** and **y=5**, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.
Given that **x=5**, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	is equal to	x==8	*false*
		x==5	*true*
===	is exactly equal to (value and type)	x==="5"	*false*
		x===5	*true*
!=	is not equal	x!=8	*true*
!==	is not equal (neither value nor type)	x!=="5"	*true*
		x!==5	*false*

>	is greater than	x>8	*false*
<	is less than	x<8	*true*
>=	is greater than or equal to	x>=8	*false*
<=	is less than or equal to	x<=8	*true*

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

if (age<18) x="Too young";

You will learn more about the use of conditional statements in the next chapter of this tutorial.

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x=6 and y=3**, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

JavaScript If...Else Statements

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

---

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
{
  code to be executed if condition is true
}
```

Note that *if* is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example

Make a "Good day" greeting if the time is less than 20:00:

```
if (time<20)
{
  x="Good day";
}
```

The result of *x* will be:
Good day

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
{
  code to be executed if condition is true
}
else
{
  code to be executed if condition is not true
}
```

Example

If the time is less than 20:00, you will get a "Good day" greeting, otherwise you will get a "Good evening" greeting

```
if (time<20)
{
  x="Good day";
}
else
{
  x="Good evening";
}
```

The result of *x* will be:
Good day

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
{
  code to be executed if condition1 is true
}
else if (condition2)
{
  code to be executed if condition2 is true
}
else
{
  code to be executed if neither condition1 nor condition2 is true
}
```

Example

If the time is less than 10:00, you will get a "Good morning" greeting, if not, but the time is less than 20:00, you will get a "Good day" greeting, otherwise you will get a "Good evening" greeting:

```
if (time<10)
{
  x="Good morning";
}
else if (time<20)
{
  x="Good day";
}
else
{
  x="Good evening";
}
```

The result of *x* will be:

Good day

JavaScript Switch Statement

The switch statement is used to perform different action based on different conditions.

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

Example

Display today's weekday-name. Note that Sunday=0, Monday=1, Tuesday=2, etc:

```
var day=new Date().getDay();
switch (day)
{
case 0:
  x="Today it's Sunday";
  break;
case 1:
  x="Today it's Monday";
  break;
case 2:
  x="Today it's Tuesday";
  break;
case 3:
  x="Today it's Wednesday";
  break;
case 4:
  x="Today it's Thursday";
  break;
case 5:
  x="Today it's Friday";
  break;
case 6:
  x="Today it's Saturday";
  break;
}
```

The result of *x* will be:
Today it's Wednesday

The *default* Keyword

Use the *default* keyword to specify what to do if there is no match:

Example
If it is NOT Saturday or Sunday, then write a default message:

```
var day=new Date().getDay();
switch (day)
{
case 6:
  x="Today it's Saturday";
  break;
case 0:
  x="Today it's Sunday";
  break;
default:
  x="Looking forward to the Weekend";
}
```

The result of *x* will be:
Looking forward to the Weekend

JavaScript For Loop

Loops can execute a block of code a number of times.

JavaScript Loops
Loops are handy, if you want to run the same code over and over again, each time with a different value.
Often this is the case when working with arrays:
Instead of writing:

```
document.write(cars[0] + "<br>");
document.write(cars[1] + "<br>");
document.write(cars[2] + "<br>");
document.write(cars[3] + "<br>");
document.write(cars[4] + "<br>");
document.write(cars[5] + "<br>");
```

You can write:

```
for (var i=0;i<cars.length;i++)
{
    document.write(cars[i] + "<br>");
}
```

Different Kinds of Loops
JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

The For Loop

The for loop is often the tool you will use when you want to create a loop.

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3)
{
    the code block to be executed
}
```

Statement 1 is executed before the loop (the code block) starts.

Statement 2 defines the condition for running the loop (the code block).

Statement 3 is executed each time after the loop (the code block) has been executed.

Example

```
for (var i=0; i<5; i++)
{
    x=x + "The number is " + i + "<br>";
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i=0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

JavaScript While Loop

Loops can execute a block of code as long as a specified condition is true.

The While Loop

The while loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition)
{
    code block to be executed
}
```

Example

The loop in this example will continue to run as long as the variable i is less than 5:

Example

```
while (i<5)
{
    x=x + "The number is " + i + "<br>";
    i++;
}
```

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do
{
    code block to be executed
}
while (condition);
```

Example

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do
{
    x=x + "The number is " + i + "<br>";
    i++;
}
while (i<5);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a **for loop** to display all the values in the cars array:

Example

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
for (;cars[i];)
{
   document.write(cars[i] + "<br>");
   i++;
}
```

The loop in this example uses a **while loop** to display all the values in the cars array:

Example

```
cars=["BMW","Volvo","Saab","Ford"];
var i=0;
while (cars[i])
{
   document.write(cars[i] + "<br>");
   i++;
}
```