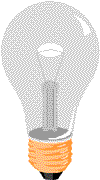
1.Js Introduction

Attribute Values

In this example JavaScript changes the value of the src (source) attribute of an <img> tag:

The Light Bulb

Turn on the light  Turn off the light

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb)

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

document.getElementById("demo").style.fontSize = "35px";  
or  
document.getElementById('demo').style.fontSize = '35px';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_style)

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

Example

document.getElementById("demo").style.display = "none";  
or  
document.getElementById('demo').style.display = 'none';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_hide)

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the display style:

Example

document.getElementById("demo").style.display = "block";  
or  
document.getElementById('demo').style.display = 'block';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_show)

2.JS Where To

## The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto)

Old JavaScript examples may use a type attribute: <script type="text/javascript">.  
The type attribute is not required. JavaScript is the default scripting language in HTML.

## JavaScript Functions and Events

A JavaScript **function** is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an **event** occurs, like when the user clicks a button.

You will learn much more about functions and events in later chapters.

## JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

## JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

### Example

<!DOCTYPE html>  
<html>

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

<body>

<h1>A Web Page</h1>  
<p id="demo">A Paragraph</p>  
<button type="button" onclick="myFunction()">Try it</button>

</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_head)

## JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

### Example

<!DOCTYPE html>  
<html>  
<body>   
  
<h1>A Web Page</h1>  
<p id="demo">A Paragraph</p>  
<button type="button" onclick="myFunction()">Try it</button>  
  
<script>  
function myFunction() {  
   document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_body)

Placing scripts at the bottom of the <body> element improves the display speed, because script compilation slows down the display.

## External JavaScript

Scripts can also be placed in external files:

### External file: myScript.js

function myFunction() {  
   document.getElementById("demo").innerHTML = "Paragraph changed.";  
}

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension**.js**.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

### Example

<script src="myScript.js"></script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_external)

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

External scripts cannot contain <script> tags.

## External JavaScript Advantages

Placing scripts in external files has some advantages:

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

To add several script files to one page  - use several script tags:

### Example

<script src="myScript1.js"></script>  
<script src="myScript2.js"></script>

## External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

### Example

<script src="https://www.w3schools.com/js/myScript1.js"></script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_url)

This example uses a script located in a specified folder on the current web site:

### Example

<script src="/js/myScript1.js"></script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_url_relative)

This example links to a script located in the same folder as the current page:

### Example

<script src="myScript1.js"></script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_external)

3.JS Output

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

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

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My First Paragraph</p>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = 5 + 6;  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_dom)

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
document.write(5 + 6);  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write)

Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**:

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<button type="button" onclick="document.write(5 + 6)">Try it</button>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write_over)

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
window.alert(5 + 6);  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_alert)

Using console.log()

For debugging purposes, you can use the **console.log()** method to display data.

You will learn more about debugging in a later chapter.

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<script>  
console.log(5 + 6);  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_console)

4.JS Statements

### Example

var x, y, z;    // Statement 1  
x = 5;          // Statement 2  
y = 6;          // Statement 3  
z = x + y;      // Statement 4

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements)

## JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

In HTML, JavaScript programs are executed by the web browser.

## JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

### Example

document.getElementById("demo").innerHTML = "Hello Dolly.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statement)

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

## Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

var a, b, c;     // Declare 3 variables  
a = 5;           // Assign the value 5 to a  
b = 6;           // Assign the value 6 to b  
c = a + b;       // Assign the sum of a and b to c

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon1)

When separated by semicolons, multiple statements on one line are allowed:

a = 5; b = 6; c = a + b;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon2)

On the web, you might see examples without semicolons.   
Ending statements with semicolon is not required, but highly recommended.

## JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

var person = "Hege";  
var person="Hege";

A good practice is to put spaces around operators ( = + - \* / ):

var x = y + z;

## JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

### Example

document.getElementById("demo").innerHTML =  
"Hello Dolly!";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_linebreak)

## JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

### Example

function myFunction() {  
    document.getElementById("demo1").innerHTML = "Hello Dolly!";  
    document.getElementById("demo2").innerHTML = "How are you?";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_blocks)

In this tutorial we use 4 spaces of indentation for code blocks.  
You will learn more about functions later in this tutorial.

## JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| break | Terminates a switch or a loop |
| continue | Jumps out of a loop and starts at the top |
| debugger | Stops the execution of JavaScript, and calls (if available) the debugging function |
| do ... while | Executes a block of statements, and repeats the block, while a condition is true |
| for | Marks a block of statements to be executed, as long as a condition is true |
| function | Declares a function |
| if ... else | Marks a block of statements to be executed, depending on a condition |
| return | Exits a function |
| switch | Marks a block of statements to be executed, depending on different cases |
| try ... catch | Implements error handling to a block of statements |
| var | Declares a variable |

5.JS Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

var x, y;          // How to declare variables  
x = 5; y = 6;      // How to assign values  
z = x + y;         // How to compute values

## JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

## JavaScript Literals

The most important rules for writing fixed values are:

**Numbers** are written with or without decimals:

10.50  
  
1001

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_numbers)

**Strings** are text, written within double or single quotes:

"John Doe"  
  
'John Doe'

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_strings)

## JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var**keyword to **declare** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

var x;  
  
x = 6;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_variables)

## JavaScript Operators

JavaScript uses **arithmetic operators** ( + - \*  / ) to **compute** values:

(5 + 6) \* 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_operators)

JavaScript uses an **assignment operator** ( = ) to **assign** values to variables:

var x, y;  
x = 5;  
y = 6;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_assign)

## JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, 5 \* 10 evaluates to 50:

5 \* 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions)

Expressions can also contain variable values:

x \* 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions_variables)

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe":

"John" + " " + "Doe"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions_strings)

## JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The **var** keyword tells the browser to create variables:

var x, y;  
x = 5 + 6;  
y = x \* 10;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_keywords)

## JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes **//** or between **/\*** and **\*/** is treated as a **comment**.

Comments are ignored, and will not be executed:

var x = 5;   // I will be executed  
  
// var x = 6;   I will NOT be executed

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_comments)

You will learn more about comments in a later chapter.

## JavaScript Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, or an underscore (\_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

Numbers are not allowed as the first character.  
This way JavaScript can easily distinguish identifiers from numbers.

## JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

var lastname, lastName;  
lastName = "Doe";  
lastname = "Peterson";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_case)

JavaScript does not interpret **VAR** or **Var** as the keyword **var**.

## JavaScript and Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name:

**Hyphens:**

first-name, last-name, master-card, inter-city.

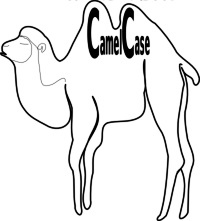
Hyphens are not allowed in JavaScript. They are reserved for subtractions.

**Underscore:**

first\_name, last\_name, master\_card, inter\_city.

**Upper Camel Case (Pascal Case):**

FirstName, LastName, MasterCard, InterCity.



**Lower Camel Case:**

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

## JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

For a closer look, please study our [Complete Unicode Reference](https://www.w3schools.com/charsets/ref_html_utf8.asp).

6.JS comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

## Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

### Example

// Change heading:  
document.getElementById("myH").innerHTML = "My First Page";  
// Change paragraph:  
document.getElementById("myP").innerHTML = "My first paragraph.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments1)

This example uses a single line comment at the end of each line to explain the code:

### Example

var x = 5;      // Declare x, give it the value of 5  
var y = x + 2;  // Declare y, give it the value of x + 2

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments5)

## Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

### Example

/\*  
The code below will change  
the heading with id = "myH"  
and the paragraph with id = "myP"  
in my web page:  
\*/  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments2)

It is most common to use single line comments.  
Block comments are often used for formal documentation.

## Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding // in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

### Example

//document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments3)

This example uses a comment block to prevent execution of multiple lines:

### Example

/\*  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
\*/

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_comments4)

7.JS Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

Example

var x = 5;  
var y = 6;  
var z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables)

From the example above, you can expect:

* x stores the value 5
* y stores the value 6
* z stores the value 11

Much Like Algebra

In this example, price1, price2, and total, are variables:

Example

var price1 = 5;  
var price2 = 6;  
var total = price1 + price2;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_total)

In programming, just like in algebra, we use variables (like price1) to hold values.

In programming, just like in algebra, we use variables in expressions (total = price1 + price2).

From the example above, you can calculate the total to be 11.

JavaScript variables are containers for storing data values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

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

* Names can contain letters, digits, underscores, and dollar signs.
* Names must begin with a letter
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case sensitive (y and Y are different variables)
* Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

The Assignment Operator

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

This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

The "equal to" operator is written like == in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

var pi = 3.14;  
var person = "John Doe";  
var answer = 'Yes I am!';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_types)

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the **var** keyword:

var carName;

After the declaration, the variable has no value. (Technically it has the value of **undefined**)

To **assign** a value to the variable, use the equal sign:

carName = "Volvo";

You can also assign a value to the variable when you declare it:

var carName = "Volvo";

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

<p id="demo"></p>  
  
<script>  
var carName = "Volvo";  
document.getElementById("demo").innerHTML = carName;   
</script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_create)

It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with **var** and separate the variables by **comma**:

var person = "John Doe", carName = "Volvo", price = 200;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multi)

A declaration can span multiple lines:

var person = "John Doe",  
carName = "Volvo",  
price = 200;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multiline)

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value **undefined**.

The variable carName will have the value undefined after the execution of this statement:

Example

var carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_undefined)

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

Example

var carName = "Volvo";  
var carName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_redefine)

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

Example

var x = 5 + 2 + 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_numbers)

You can also add strings, but strings will be concatenated:

Example

var x = "John" + " " + "Doe";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_strings)

Also try this:

Example

var x = "5" + 2 + 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_string_number)

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

Example

var x = 2 + 3 + "5";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_number_string)

8.Js Operators

### Example

Assign values to variables and add them together:

var x = 5;         // assign the value 5 to x  
var y = 2;         // assign the value 2 to y  
var z = x + y;     // assign the value 7 to z (x + y)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper)

The **assignment** operator (=) assigns a value to a variable.

### Assignment

var x = 10;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_equal)

The **addition** operator (+) adds numbers:

### Adding

var x = 5;  
var y = 2;  
var z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add)

The **multiplication** operator (\*) multiplies numbers.

### Multiplying

var x = 5;  
var y = 2;  
var z = x \* y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult)

## JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

Arithmetic operators are fully described in the [**JS Arithmetic**](https://www.w3schools.com/js/js_arithmetic.asp) chapter.

## JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

The **addition assignment** operator (+=) adds a value to a variable.

### Assignment

var x = 10;  
x += 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_plusequal)

Assignment operators are fully described in the [**JS Assignment**](https://www.w3schools.com/js/js_assignment.asp) chapter.

## JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

### Example

var txt1 = "John";  
var txt2 = "Doe";  
var txt3 = txt1 + " " + txt2;

The result of txt3 will be:

John Doe

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concatenate)

The += assignment operator can also be used to add (concatenate) strings:

### Example

var txt1 = "What a very ";  
txt1 += "nice day";

The result of txt1 will be:

What a very nice day

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat4)

When used on strings, the + operator is called the concatenation operator.

## Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

### Example

var x = 5 + 5;  
var y = "5" + 5;  
var z = "Hello" + 5;

The result of x, y, and z will be:

10  
55  
Hello5

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat5)

If you add a number and a string, the result will be a string!

## JavaScript Comparison Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

Comparison operators are fully described in the [**JS Comparisons**](https://www.w3schools.com/js/js_comparisons.asp) chapter.

## JavaScript Logical Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | logical and |
| || | logical or |
| ! | logical not |

Logical operators are fully described in the [**JS Comparisons**](https://www.w3schools.com/js/js_comparisons.asp) chapter.

## JavaScript Type Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

Type operators are fully described in the [**JS Type Conversion**](https://www.w3schools.com/js/js_type_conversion.asp) chapter.

## JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | Zero fill left shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | Signed right shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | Zero fill right shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

9.JS Arithmetic

## JavaScript Arithmetic Operators

Arithmetic operators perform arithmetic on numbers (literals or variables).

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus (Remainder) |
| ++ | Increment |
| -- | Decrement |

## Arithmetic Operations

A typical arithmetic operation operates on two numbers.

The two numbers can be literals:

### Example

var x = 100 + 50;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_operation)

or variables:

### Example

var x = a + b;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_variables)

or expressions:

### Example

var x = (100 + 50) \* a;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_expressions)

## Operators and Operands

The numbers (in an arithmetic operation) are called **operands**.

The operation (to be performed between the two operands) is defined by an **operator**.

|  |  |  |
| --- | --- | --- |
| **Operand** | **Operator** | **Operand** |
| 100 | + | 50 |

## Adding

The **addition** operator (+) adds numbers:

### Example

var x = 5;  
var y = 2;  
var z = x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add)

## Subtracting

The **subtraction** operator (-) subtracts numbers.

### Example

var x = 5;  
var y = 2;  
var z = x - y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_sub)

## Multiplying

The **multiplication** operator (\*) multiplies numbers.

### Example

var x = 5;  
var y = 2;  
var z = x \* y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult)

## Dividing

The **division** operator (/) divides numbers.

### Example

var x = 5;  
var y = 2;  
var z = x / y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_div)

## Remainder

The **modulus** operator (%) returns the division remainder.

### Example

var x = 5;  
var y = 2;  
var z = x % y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mod)

In arithmetic, the division of two integers produces a **quotient** and a **remainder**.  
In mathematics, the result of a **modulo operation** is the **remainder** of an arithmetic division.

## Incrementing

The **increment** operator (++) increments numbers.

### Example

var x = 5;  
x++;  
var z = x;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_increment)

## Decrementing

The **decrement** operator (--) decrements numbers.

### Example

var x = 5;  
x--;  
var z = x;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_decrement)

## Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

### Example

var x = 100 + 50 \* 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence1)

Is the result of example above the same as 150 \* 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (\*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

And (as in school mathematics) the precedence can be changed by using parentheses:

### Example

var x = (100 + 50) \* 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence2)

When using parentheses, the operations inside the parentheses are computed first.

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

### Example

var x = 100 + 50 - 3;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence3)

## JavaScript Operator Precedence Values

Pale red entries indicates ECMAScript 2015 (ES6) or higher.

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Operator** | **Description** | **Example** |
| 20 | ( ) | Expression grouping | (3 + 4) |
|  |  |  |  |
| 19 | . | Member | person.name |
| 19 | [] | Member | person["name"] |
| 19 | () | Function call | myFunction() |
| 19 | new | Create | new Date() |
|  |  |  |  |
| 17 | ++ | Postfix Increment | i++ |
| 17 | -- | Postfix Decrement | i-- |
|  |  |  |  |
| 16 | ++ | Prefix Increment | ++i |
| 16 | -- | Prefix Decrement | --i |
| 16 | ! | Logical not | !(x==y) |
| 16 | typeof | Type | typeof x |
|  |  |  |  |
| 15 | \*\* | Exponentiation (ES7) | 10 \*\* 2 |
|  |  |  |  |
| 14 | \* | Multiplication | 10 \* 5 |
| 14 | / | Division | 10 / 5 |
| 14 | % | Division Remainder | 10 % 5 |
|  |  |  |  |
| 13 | + | Addition | 10 + 5 |
| 13 | - | Subtraction | 10 - 5 |
|  |  |  |  |
| 12 | << | Shift left | x << 2 |
| 12 | >> | Shift right | x >> 2 |
| 12 | >>> | Shift right (unsigned) | x >>> 2 |
|  |  |  |  |
| 11 | < | Less than | x < y |
| 11 | <= | Less than or equal | x <= y |
| 11 | > | Greater than | x > y |
| 11 | >= | Greater than or equal | x >= y |
| 11 | in | Property in Object | "PI" in Math |
| 11 | instanceof | Instance of Object | instanceof Array |
|  |  |  |  |
| 10 | == | Equal | x == y |
| 10 | === | Strict equal | x === y |
| 10 | != | Unequal | x != y |
| 10 | !== | Strict unequal | x !== y |
|  |  |  |  |
| 9 | & | Bitwise AND | x & y |
| 8 | ^ | Bitwise XOR | x ^ y |
| 7 | | | Bitwise OR | x | y |
| 6 | && | Logical AND | x && y |
| 5 | || | Logical OR | x || y |
| 4 | ? : | Condition | ? "Yes" : "No" |
|  |  |  |  |
| 3 | += | Assignment | x += y |
| 3 | += | Assignment | x += y |
| 3 | -= | Assignment | x -= y |
| 3 | \*= | Assignment | x \*= y |
| 3 | %= | Assignment | x %= y |
| 3 | <<= | Assignment | x <<= y |
| 3 | >>= | Assignment | x >>= y |
| 3 | >>>= | Assignment | x >>>= y |
| 3 | &= | Assignment | x &= y |
| 3 | ^= | Assignment | x ^= y |
| 3 | |= | Assignment | x |= y |
|  |  |  |  |
| 2 | yield | Pause Function | yield x |
| 1 | , | Comma | 5 , 6 |

Expressions in parentheses are fully computed before the value is used in the rest of the expression.

10.JS Assignment

## JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| <<= | x <<= y | x = x << y |
| >>= | x >>= y | x = x >> y |
| >>>= | x >>>= y | x = x >>> y |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| |= | x |= y | x = x | y |
| \*\*= | x \*\*= y | x = x \*\* y |

The \*\*= operator is an experimental part of the ECMAScript 2016 proposal (ES7). It is not stable across browsers. Do not use it.

## Assignment Examples

The = assignment operator assigns a value to a variable.

### Assignment

var x = 10;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_equal)

The += assignment operator adds a value to a variable.

### Assignment

var x = 10;  
x += 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_plusequal)

The -= assignment operator subtracts a value from a variable.

### Assignment

var x = 10;  
x -= 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_minequal)

The \*= assignment operator multiplies a variable.

### Assignment

var x = 10;  
x \*= 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_multequal)

The /= assignment divides a variable.

### Assignment

var x = 10;  
x /= 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_divequal)

The %= assignment operator assigns a remainder to a variable.

### Assignment

var x = 10;  
x %= 5;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_modequal)

11.JS Data Types

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

var length = 16;                               // Number  
var lastName = "Johnson";                      // String  
var x = {firstName:"John", lastName:"Doe"};    // Object

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

var x = 16 + "Volvo";

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

var x = "16" + "Volvo";

When adding a number and a string, JavaScript will treat the number as a string.

Example

var x = 16 + "Volvo";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstring)

Example

var x = "Volvo" + 16;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstring2)

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

JavaScript:

var x = 16 + 4 + "Volvo";

Result:

20Volvo

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_1)

JavaScript:

var x = "Volvo" + 16 + 4;

Result:

Volvo164

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_2)

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

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

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_dynamic)

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

var carName = "Volvo XC60";   // Using double quotes  
var carName = 'Volvo XC60';   // Using single quotes

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string_quotes)

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

var answer = "It's alright";             // Single quote inside double quotes  
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes  
var answer = 'He is called "Johnny"';    // Double quotes inside single quotes

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string)

You will learn more about strings later in this tutorial.

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

var x1 = 34.00;     // Written with decimals  
var x2 = 34;        // Written without decimals

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers)

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

var y = 123e5;      // 12300000  
var z = 123e-5;     // 0.00123

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers_large)

You will learn more about numbers later in this tutorial.

JavaScript Booleans

Booleans can only have two values: true or false.

Example

var x = 5;  
var y = 5;  
var z = 6;  
(x == y)       // Returns true  
(x == z)       // Returns false

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_bolean)

Booleans are often used in conditional testing.

You will learn more about conditional testing later in this tutorial.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example

var cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_array)

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about arrays later in this tutorial.

JavaScript Objects

JavaScript objects are written with curly braces.

Object properties are written as name:value pairs, separated by commas.

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_object)

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about objects later in this tutorial.

The typeof Operator

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable.

The **typeof** operator returns the type of a variable or an expression:

Example

typeof ""                  // Returns "string"  
typeof "John"              // Returns "string"  
typeof "John Doe"          // Returns "string"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_string)

Example

typeof 0                   // Returns "number"  
typeof 314                 // Returns "number"  
typeof 3.14                // Returns "number"  
typeof (3)                 // Returns "number"  
typeof (3 + 4)             // Returns "number"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_number)

Undefined

In JavaScript, a variable without a value, has the value**undefined**. The typeof is also **undefined**.

Example

var car;                // Value is undefined, type is undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined)

Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

Example

car = undefined;        // Value is undefined, type is undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_2)

Empty Values

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

Example

var car = "";              // The value is "", the typeof is "string"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_empty)

Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

You can consider it a bug in JavaScript that typeof null is an object. It should be null.

You can empty an object by setting it to null:

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
person = null;        // Now value is null, but type is still an object

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_null)

You can also empty an object by setting it to undefined:

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
person = undefined;   // Now both value and type is undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_1)

Difference Between Undefined and Null

Undefined and null are equal in value but different in type:

typeof undefined           // undefined  
typeof null                // object  
  
null === undefined         // false  
null == undefined          // true

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_3)

Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

The **typeof** operator can return one of these primitive types:

* string
* number
* boolean
* undefined

Example

typeof "John"              // Returns "string"   
typeof 3.14                // Returns "number"  
typeof true                // Returns "boolean"  
typeof false               // Returns "boolean"  
typeof x                   // Returns "undefined" (if x has no value)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_primitive)

Complex Data

The **typeof** operator can return one of two complex types:

* function
* object

The typeof operator returns object for both objects, arrays, and null.

The typeof operator does not return object for functions.

Example

typeof {name:'John', age:34} // Returns "object"  
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)  
typeof null                  // Returns "object"  
typeof function myFunc(){}   // Returns "function"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof_complex)

12.JS Function

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

function myFunction(p1, p2) {  
    return p1 \* p2;              // The function returns the product of p1 and p2  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_functions)

JavaScript Function Syntax

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:  
**(*parameter1, parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

function *name*(*parameter1, parameter2, parameter3*) {  
    *code to be executed*  
}

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

* When an event occurs (when a user clicks a button)
* When it is invoked (called) from JavaScript code
* Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return

When JavaScript reaches a **return statement**, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

var x = myFunction(4, 3);    // Function is called, return value will end up in x  
  
function myFunction(a, b) {  
    return a \* b;            // Function returns the product of a and b  
}

The result in x will be:

12

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_return)

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Example

Convert Fahrenheit to Celsius:

function toCelsius(fahrenheit) {  
    return (5/9) \* (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius(77);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius)

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function definition instead of the function result:

Example

function toCelsius(fahrenheit) {  
    return (5/9) \* (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius_2)

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

var x = toCelsius(77);  
var text = "The temperature is " + x + " Celsius";

You can use the function directly, as a variable value:

var text = "The temperature is " + toCelsius(77) + " Celsius";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_variable)

You will learn a lot more about functions later in this tutorial.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

// code here can NOT use carName  
  
function myFunction() {  
    var carName = "Volvo";  
    // code here CAN use carName  
}  
  
// code here can NOT use carName

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_scope)

13.JS Objects

## Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

|  |  |  |
| --- | --- | --- |
| **Object** | **Properties** | **Methods** |
| https://www.w3schools.com/js/objectExplained.gif | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()   car.stop() |

All cars have the same **properties**, but the property **values** differ from car to car.

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

## JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

var car = "Fiat";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_variable)

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

var car = {type:"Fiat", model:"500", color:"white"};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_object)

The values are written as **name:value** pairs (name and value separated by a colon).

JavaScript objects are containers for **named values**called properties or methods.

## Object Definition

You define (and create) a JavaScript object with an object literal:

### Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_1)

Spaces and line breaks are not important. An object definition can span multiple lines:

### Example

var person = {  
    firstName:"John",  
    lastName:"Doe",  
    age:50,  
    eyeColor:"blue"  
};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_2)

## Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |

## Accessing Object Properties

You can access object properties in two ways:

*objectName.propertyName*

or

*objectName["propertyName"]*

### Example1

person.lastName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_1)

### Example2

person["lastName"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_2)

## Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

A method is a function stored as a property.

### Example

var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id       : 5566,  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};

## The ****this**** Keyword

In a function definition, **this** refers to the "owner" of the function.

In the example above, **this** is the **person object** that "owns" the **fullName** function.

In other words, **this.firstName** means the **firstName** property of **this object**.

Read more about the **this** keyword at [JS this Keyword](https://www.w3schools.com/js/js_this.asp).

## Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

### Example

name = person.fullName();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_method)

If you access a method **without** the () parentheses, it will return the **function definition**:

### Example

name = person.fullName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_function)

## Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

var x = new String();        // Declares x as a String object  
var y = new Number();        // Declares y as a Number object  
var z = new Boolean();       // Declares z as a Boolean object

14.Js Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

* An HTML web page has finished loading
* An HTML input field was changed
* An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

<*element* *event*=**'*some JavaScript*'**>

With double quotes:

<*element* *event*=**"*some JavaScript*"**>

In the following example, an onclick attribute (with code), is added to a button element:

Example

<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick1)

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **this**.innerHTML):

Example

<button onclick="this.innerHTML = Date()">The time is?</button>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick)

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

<button onclick="displayDate()">The time is?</button>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_events1)

Common HTML Events

Here is a list of some common HTML events:

|  |  |
| --- | --- |
| **Event** | **Description** |
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

The list is much longer: [W3Schools JavaScript Reference HTML DOM Events](https://www.w3schools.com/jsref/dom_obj_event.asp).

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

* Things that should be done every time a page loads
* Things that should be done when the page is closed
* Action that should be performed when a user clicks a button
* Content that should be verified when a user inputs data
* And more ...

Many different methods can be used to let JavaScript work with events:

* HTML event attributes can execute JavaScript code directly
* HTML event attributes can call JavaScript functions
* You can assign your own event handler functions to HTML elements
* You can prevent events from being sent or being handled
* And more ...

You will learn a lot more about events and event handlers in the HTML DOM chapters.

15.JS Strings

JavaScript strings are used for storing and manipulating text.

## JavaScript Strings

A JavaScript string is zero or more characters written inside quotes.

### Example

var x = "John Doe";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string)

You can use single or double quotes:

### Example

var carname = "Volvo XC60";  // Double quotes  
var carname = 'Volvo XC60';  // Single quotes

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_quotes)

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

### Example

var answer = "It's alright";  
var answer = "He is called 'Johnny'";  
var answer = 'He is called "Johnny"';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_quotes_mixed)

## String Length

The length of a string is found in the built in property **length**:

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_length)

## Special Characters

Because strings must be written within quotes, JavaScript will misunderstand this string:

var x = "We are the so-called "Vikings" from the north.";

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Description** |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

The sequence **\"**  inserts a double quote in a string:

### Example

var x = "We are the so-called \"Vikings\" from the north.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_escape_quotes2)

The sequence **\'**  inserts a single quote in a string:

### Example

var x = 'It\'s alright.';

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_escape_quotes1)

The sequence **\\**  inserts a backslash in a string:

### Example

var x = "The character \\ is called backslash.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_escape_backslash)

Six other escape sequences are valid in JavaScript:

|  |  |
| --- | --- |
| **Code** | **Result** |
| \b | Backspace |
| \f | Form Feed |
| \n | New Line |
| \r | Carriage Return |
| \t | Horizontal Tabulator |
| \v | Vertical Tabulator |

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

## Breaking Long Code Lines

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

### Example

document.getElementById("demo").innerHTML =  
"Hello Dolly!";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_linebreak)

You can also break up a code line **within a text string** with a single backslash:

### Example

document.getElementById("demo").innerHTML = "Hello \  
Dolly!";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_break)

The \ method is not the preferred method. It might not have universal support.  
Some browsers do not allow spaces behind the \ character.

A safer way to break up a string, is to use string addition:

### Example

document.getElementById("demo").innerHTML = "Hello " +   
"Dolly!";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_break_ok)

You cannot break up a code line with a backslash:

### Example

document.getElementById("demo").innerHTML = \   
"Hello Dolly!";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_codebreak)

## Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

**var firstName = "John";**

But strings can also be defined as objects with the keyword new:

**var firstName = new String("John");**

### Example

var x = "John";  
var y = new String("John");  
  
// typeof x will return string  
// typeof y will return object

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object)

Don't create strings as objects. It slows down execution speed.  
The **new** keyword complicates the code. This can produce some unexpected results:

When using the == operator, equal strings are equal:

### Example

var x = "John";               
var y = new String("John");  
  
// (x == y) is true because x and y have equal values

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object1)

When using the === operator, equal strings are not equal, because the === operator expects equality in both type and value.

### Example

var x = "John";               
var y = new String("John");  
  
// (x === y) is false because x and y have different types (string and object)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object2)

Or even worse. Objects cannot be compared:

### Example

var x = new String("John");               
var y = new String("John");  
  
// (x == y) is false because x and y are different objects

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object3)

### Example

var x = new String("John");               
var y = new String("John");  
  
// (x === y) is false because x and y are different objects

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object4)

16.JS String Methods

String methods help you to work with strings.

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The **length** property returns the length of a string:

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_length)

Finding a String in a String

The **indexOf()** method returns the index of (the position of) the **first** occurrence of a specified text in a string:

Example

var str = "Please locate where 'locate' occurs!";  
var pos = str.indexOf("locate");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_indexof)

JavaScript counts positions from zero.  
0 is the first position in a string, 1 is the second, 2 is the third ...

The **lastIndexOf()** method returns the index of the **last** occurrence of a specified text in a string:

Example

var str = "Please locate where 'locate' occurs!";  
var pos = str.lastIndexOf("locate");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_lastindexof)

Both indexOf(), and lastIndexOf() return -1 if the text is not found.

Example

var str = "Please locate where 'locate' occurs!";  
var pos = str.lastIndexOf("John");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_indexof_1)

Both methods accept a second parameter as the starting position for the search:

Example

var str = "Please locate where 'locate' occurs!";  
var pos = str.indexOf("locate",15);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_indexof_2)

Searching for a String in a String

The **search()** method searches a string for a specified value and returns the position of the match:

Example

var str = "Please locate where 'locate' occurs!";  
var pos = str.search("locate");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_search_locate)

Did You Notice?

The two methods, indexOf() and search(), are **equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are **NOT** equal. These are the differences:

* The search() method cannot take a second start position argument.
* The indexOf() method cannot take powerful search values (regular expressions).

You will learn more about regular expressions in a later chapter.

Extracting String Parts

There are 3 methods for extracting a part of a string:

* slice(start, end)
* substring(start, end)
* substr(start, length)

The slice() Method

**slice()** extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the starting index (position), and the ending index (position).

This example slices out a portion of a string from position 7 to position 13:

Example

var str = "Apple, Banana, Kiwi";  
var res = str.slice(7, 13);

The result of res will be:

Banana

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice)

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

Example

var str = "Apple, Banana, Kiwi";  
var res = str.slice(-12, -6);

The result of res will be:

Banana

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice_negative)

If you omit the second parameter, the method will slice out the rest of the string:

Example

var res = str.slice(7);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice_rest)

or, counting from the end:

Example

var res = str.slice(-12);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice_rest_negative)

Negative positions do not work in Internet Explorer 8 and earlier.

The substring() Method

**substring()** is similar to slice().

The difference is that substring() cannot accept negative indexes.

Example

var str = "Apple, Banana, Kiwi";  
var res = str.substring(7, 13);

The result of *res* will be:

Banana

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substring)

If you omit the second parameter, substring() will slice out the rest of the string.

The substr() Method

**substr()** is similar to slice().

The difference is that the second parameter specifies the **length** of the extracted part.

Example

var str = "Apple, Banana, Kiwi";  
var res = str.substr(7, 6);

The result of res will be:

Banana

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substr)

If you omit the second parameter, substr() will slice out the rest of the string.

Example

var str = "Apple, Banana, Kiwi";  
var res = str.substr(7);

The result of res will be:

Banana, Kiwi

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substr1)

If the first parameter is negative, the position counts from the end of the string.

Example

var str = "Apple, Banana, Kiwi";  
var res = str.substr(-4);

The result of res will be:

Kiwi

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substr2)

Replacing String Content

The **replace()** method replaces a specified value with another value in a string:

Example

str = "Please visit Microsoft!";  
var n = str.replace("Microsoft", "W3Schools");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace)

The replace() method does not change the string it is called on. It returns a new string.

By default, the replace() function replaces **only the first** match:

Example

str = "Please visit Microsoft and Microsoft!";  
var n = str.replace("Microsoft", "W3Schools");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace_first)

By default, the replace() function is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example

str = "Please visit Microsoft!";  
var n = str.replace("MICROSOFT", "W3Schools");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace_case)

To replace case insensitive, use a **regular expression** with an **/i** flag (insensitive):

Example

str = "Please visit Microsoft!";  
var n = str.replace(/MICROSOFT/i, "W3Schools");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace_insensitive)

Note that regular expressions are written without quotes.

To replace all matches, use a **regular expression** with a **/g** flag (global match):

Example

str = "Please visit Microsoft and Microsoft!";  
var n = str.replace(/Microsoft/g, "W3Schools");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace_global)

You will learn a lot more about regular expressions in the chapter [JavaScript Regular Expressions](https://www.w3schools.com/js/js_regexp.asp).

Converting to Upper and Lower Case

A string is converted to upper case with **toUpperCase()**:

Example

var text1 = "Hello World!";       // String  
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_toupper)

A string is converted to lower case with **toLowerCase()**:

Example

var text1 = "Hello World!";       // String  
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_tolower)

The concat() Method

**concat()** joins two or more strings:

Example

var text1 = "Hello";  
var text2 = "World";  
var text3 = text1.concat(" ", text2);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_concat)

The **concat()** method can be used instead of the plus operator. These two lines do the same:

Example

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

All string methods return a new string. They don't modify the original string.  
Formally said: Strings are immutable: Strings cannot be changed, only replaced.

String.trim()

String.trim() removes whitespace from both sides of a string.

Example

var str = "       Hello World!        ";  
alert(str.trim());

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_trim)

String.trim() is not supported in Internet Explorer 8 or lower.

If you need to support IE 8, you can use String.replace with a regular expression instead:

Example

var str = "       Hello World!        ";  
alert(str.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, ''));

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_trim_regexp)

You can also use the replace solution above to add a trim function to the JavaScript String.prototype:

Example

if (!String.prototype.trim) {  
    String.prototype.trim = function () {  
    return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, '');  
};  
var str = "       Hello World!        ";  
alert(str.trim());

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_trim_polyfill)

Extracting String Characters

There are 3 methods for extracting string characters:

* charAt(position)
* charCodeAt(position)
* Property access [ ]

The charAt() Method

The **charAt()** method returns the character at a specified index (position) in a string:

Example

var str = "HELLO WORLD";  
str.charAt(0);            // returns H

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_charat)

The charCodeAt() Method

The **charCodeAt()** method returns the unicode of the character at a specified index in a string:

The method returns an UTF-16 cone integer between 0 and 65535.

Example

var str = "HELLO WORLD";  
  
str.charCodeAt(0);         // returns 72

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_charcodeat)

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

var str = "HELLO WORLD";  
str[0];                   // returns H

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_prop)

Property access might be a little **unpredictable:**

* It does not work in Internet Explorer 7 or earlier
* It makes strings look like arrays (but they are not)
* If no character is found, [ ] returns undefined, while charAt() returns an empty string.
* It is read only. str[0] = "A" gives no error (but does not work!)

Example

var str = "HELLO WORLD";  
str[0] = "A";             **// Gives no error, but does not work**  
str[0];                   // returns H

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_prop2)

If you want to work with a string as an array, you can convert it to an array.

Converting a String to an Array

A string can be converted to an array with the **split()** method:

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_split)

If the separator is omitted, the returned array will contain the whole string in index [0].

If the separator is "", the returned array will be an array of single characters:

Example

var txt = "Hello";       // String  
txt.split("");           // Split in characters

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_split_char)

Complete String Reference

17.JS Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

### Example

var x = 3.14;    // A number with decimals  
var y = 3;       // A number without decimals

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers1)

Extra large or extra small numbers can be written with scientific (exponent) notation:

### Example

var x = 123e5;    // 12300000  
var y = 123e-5;   // 0.00123

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers2)

## JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.   
  
This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

|  |  |  |
| --- | --- | --- |
| **Value (aka Fraction/Mantissa)** | **Exponent** | **Sign** |
| 52 bits (0 - 51) | 11 bits (52 - 62) | 1 bit (63) |

## Precision

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

### Example

var x = 999999999999999;   // x will be 999999999999999  
var y = 9999999999999999;  // y will be 10000000000000000

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_inaccurate1)

The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

### Example

var x = 0.2 + 0.1;         // x will be 0.30000000000000004

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_inaccurate2)

To solve the problem above, it helps to multiply and divide:

### Example

var x = (0.2 \* 10 + 0.1 \* 10) / 10;       // x will be 0.3

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_inaccurate3)

## Adding Numbers and Strings

WARNING !!

JavaScript uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

### Example

var x = 10;  
var y = 20;  
var z = x + y;           // z will be 30 (a number)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add)

If you add two strings, the result will be a string concatenation:

### Example

var x = "10";  
var y = "20";  
var z = x + y;           // z will be 1020 (a string)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings1)

If you add a number and a string, the result will be a string concatenation:

### Example

var x = 10;  
var y = "20";  
var z = x + y;           // z will be 1020 (a string)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings2)

If you add a string and a number, the result will be a string concatenation:

### Example

var x = "10";  
var y = 20;  
var z = x + y;           // z will be 1020 (a string)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings5)

A common mistake is to expect this result to be 30:

### Example

var x = 10;  
var y = 20;  
var z = "The result is: " + x + y;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings3)

A common mistake is to expect this result to be 102030:

### Example

var x = 10;  
var y = 20;  
var z = "30";  
var result = x + y + z;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_add_strings4)

The JavaScript compiler works from left to right.

First 10 + 20 is added because x and y are both numbers.

Then 30 + "30" is concatenated because z is a string.

## Numeric Strings

JavaScript strings can have numeric content:

var x = 100;         // x is a number  
  
var y = "100";       // y is a string

JavaScript will try to convert strings to numbers in all numeric operations:

This will work:

var x = "100";  
var y = "10";  
var z = x / y;       // z will be 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string1)

This will also work:

var x = "100";  
var y = "10";  
var z = x \* y;       // z will be 1000

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string2)

And this will work:

var x = "100";  
var y = "10";  
var z = x - y;       // z will be 90

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string3)

But this will not work:

var x = "100";  
var y = "10";  
var z = x + y;       // z will not be 110 (It will be 10010)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_string4)

In the last example JavaScript uses the + operator to concatenate the strings.

## NaN - Not a Number

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

### Example

var x = 100 / "Apple";  // x will be NaN (Not a Number)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_divide_string)

However, if the string contains a numeric value , the result will be a number:

### Example

var x = 100 / "10";     // x will be 10

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_divide_number)

You can use the global JavaScript function isNaN() to find out if a value is a number:

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_isnan_true)

Watch out for NaN. If you use NaN in a mathematical operation, the result will also be NaN:

### Example

var x = NaN;  
var y = 5;  
var z = x + y;         // z will be NaN

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_nan_math)

Or the result might be a concatenation:

### Example

var x = NaN;  
var y = "5";  
var z = x + y;         // z will be NaN5

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_nan_concat)

NaN is a number: typeof NaN returns number:

### Example

typeof NaN;            // returns "number"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_nan_typeof)

## Infinity

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

### Example

var myNumber = 2;  
while (myNumber != Infinity) {          // Execute until Infinity  
    myNumber = myNumber \* myNumber;  
}

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_infinity)

Division by 0 (zero) also generates Infinity:

### Example

var x =  2 / 0;          // x will be Infinity  
var y = -2 / 0;          // y will be -Infinity

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_infinity_zero)

Infinity is a number: typeof Infinity returns number.

### Example

typeof Infinity;        // returns "number"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_infinity_typeof)

## Hexadecimal

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

### Example

var x = 0xFF;           // x will be 255

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_hex)

Never write a number with a leading zero (like 07).  
Some JavaScript versions interpret numbers as octal if they are written with a leading zero.

By default, JavaScript displays numbers as **base 10** decimals.

But you can use the toString() method to output numbers from **base 2** to **base 36**.

Hexadecimal is **base 16**. Decimal is **base 10**. Octal is **base 8**. Binary is **base 2**.

### Example

var myNumber = 32;  
myNumber.toString(10);  // returns 32  
myNumber.toString(32);  // returns 10  
myNumber.toString(16);  // returns 20  
myNumber.toString(8);   // returns 40  
myNumber.toString(2);   // returns 100000

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_tostring)

## Numbers Can be Objects

Normally JavaScript numbers are primitive values created from literals:

**var x = 123;**

But numbers can also be defined as objects with the keyword new:

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

### Example

var x = 123;  
var y = new Number(123);  
  
// typeof x returns number  
// typeof y returns object

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object)

Do not create Number objects. It slows down execution speed.  
The **new** keyword complicates the code. This can produce some unexpected results:

When using the == operator, equal numbers are equal:

### Example

var x = 500;               
var y = new Number(500);  
  
// (x == y) is true because x and y have equal values

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object1)

When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

### Example

var x = 500;               
var y = new Number(500);  
  
// (x === y) is false because x and y have different types

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object2)

Or even worse. Objects cannot be compared:

### Example

var x = new Number(500);               
var y = new Number(500);  
  
// (x == y) is false because objects cannot be compared

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object3)

Note the difference between (x==y) and (x===y).  
Comparing two JavaScript objects will always return false.

18.Js Number Methods

Number methods help you work with numbers.

Number Methods and Properties

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

The toString() Method

**toString()** returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tostring)

The toExponential() Method

**toExponential()** returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

Example

var x = 9.656;  
x.toExponential(2);     // returns 9.66e+0  
x.toExponential(4);     // returns 9.6560e+0  
x.toExponential(6);     // returns 9.656000e+0

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_toexponential)

The parameter is optional. If you don't specify it, JavaScript will not round the number.

The toFixed() Method

**toFixed()** returns a string, with the number written with a specified number of decimals:

Example

var x = 9.656;  
x.toFixed(0);           // returns 10  
x.toFixed(2);           // returns 9.66  
x.toFixed(4);           // returns 9.6560  
x.toFixed(6);           // returns 9.656000

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tofixed)

toFixed(2) is perfect for working with money.

The toPrecision() Method

**toPrecision()** returns a string, with a number written with a specified length:

Example

var x = 9.656;  
x.toPrecision();        // returns 9.656  
x.toPrecision(2);       // returns 9.7  
x.toPrecision(4);       // returns 9.656  
x.toPrecision(6);       // returns 9.65600

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_toprecision)

The valueOf() Method

**valueOf()** returns a number as a number.

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_valueof)

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

All JavaScript data types have a valueOf() and a toString() method.

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

* The Number() method
* The parseInt() method
* The parseFloat() method

These methods are not **number** methods, but **global** JavaScript methods.

Global JavaScript Methods

JavaScript global methods can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

|  |  |
| --- | --- |
| **Method** | **Description** |
| Number() | Returns a number, converted from its argument. |
| parseFloat() | Parses its argument and returns a floating point number |
| parseInt() | Parses its argument and returns an integer |

The Number() Method

**Number()** can be used to convert JavaScript variables to numbers:

Example

Number(true);          // returns 1  
Number(false);         // returns 0  
Number("10");          // returns 10  
Number("  10");        // returns 10  
Number("10  ");        // returns 10  
Number(" 10  ");       // returns 10  
Number("10.33");       // returns 10.33  
Number("10,33");       // returns NaN  
Number("10 33");       // returns NaN   
Number("John");        // returns NaN

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number)

If the number cannot be converted, NaN (Not a Number) is returned.

The Number() Method Used on Dates

**Number()** can also convert a date to a number:

Example

Number(new Date("2017-09-30"));    // returns 1506729600000

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number_date)

The Number() method above returns the number of milliseconds since 1.1.1970.

The parseInt() Method

**parseInt()** parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

parseInt("10");         // returns 10  
parseInt("10.33");      // returns 10  
parseInt("10 20 30");   // returns 10  
parseInt("10 years");   // returns 10  
parseInt("years 10");   // returns NaN

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_parseint)

If the number cannot be converted, NaN (Not a Number) is returned.

The parseFloat() Method

**parseFloat()** parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

parseFloat("10");        // returns 10  
parseFloat("10.33");     // returns 10.33  
parseFloat("10 20 30");  // returns 10  
parseFloat("10 years");  // returns 10  
parseFloat("years 10");  // returns NaN

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_global_parsefloat)

If the number cannot be converted, NaN (Not a Number) is returned.

Number Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| MAX\_VALUE | Returns the largest number possible in JavaScript |
| MIN\_VALUE | Returns the smallest number possible in JavaScript |
| POSITIVE\_INFINITY | Represents infinity (returned on overflow) |
| NEGATIVE\_INFINITY | Represents negative infinity (returned on overflow) |
| NaN | Represents a "Not-a-Number" value |

JavaScript MIN\_VALUE and MAX\_VALUE

Example

var x = Number.MAX\_VALUE;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_max)

Example

var x = Number.MIN\_VALUE;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_min)

JavaScript POSITIVE\_INFINITY

Example

var x = Number.POSITIVE\_INFINITY;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_pos_infinity)

POSITIVE\_INFINITY is returned on overflow:

Example

var x = 1 / 0;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_pos_infinity2)

JavaScript NEGATIVE\_INFINITY

Example

var x = Number.NEGATIVE\_INFINITY;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_neg_infinity)

NEGATIVE\_INFINITY is returned on overflow:

Example

var x = -1 / 0;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_neg_infinity2)

JavaScript NaN - Not a Number

Example

var x = Number.NaN;

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_nan)

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

Example

var x = 100 / "Apple";  // x will be NaN (Not a Number)

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_divide_string)

Number Properties Cannot be Used on Variables

Number properties belongs to the JavaScript's number object wrapper called **Number**.

These properties can only be accessed as **Number**.MAX\_VALUE.

Using *myNumber*.MAX\_VALUE, where *myNumber* is a variable, expression, or value, will return undefined:

Example

var x = 6;  
var y = x.MAX\_VALUE;    // y becomes undefined

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_number_max_undefined)

19.JS Arrays

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

Example

var cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array)

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

var car1 = "Saab";  
var car2 = "Volvo";  
var car3 = "BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

var *array\_name* = [*item1*, *item2*, ...];

Example

var cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array)

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

var cars = [  
    "Saab",  
    "Volvo",  
    "BMW"  
];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_newlines)

Putting a comma after the last element (like "BMW",)  is inconsistent across browsers.

IE 8 and earlier will fail.

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new)

The two examples above do exactly the same. There is no need to use new Array().  
For simplicity, readability and execution speed, use the first one (the array literal method).

Access the Elements of an Array

You access an array element by referring to the **index number**.

This statement accesses the value of the first element in cars:

var name = cars[0];

Example

var cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars[0];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_element)

Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

This statement changes the value of the first element in cars:

cars[0] = "Opel";

Example

var cars = ["Saab", "Volvo", "BMW"];  
cars[0] = "Opel";  
document.getElementById("demo").innerHTML = cars[0];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_element_change)

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

var cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_full)

Arrays are Objects

Arrays are a special type of objects. The **typeof** operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, **person[0]** returns John:

Array:

var person = ["John", "Doe", 46];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_array)

Objects use **names** to access its "members". In this example, **person.firstName** returns John:

Object:

var person = {firstName:"John", lastName:"Doe", age:46};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_object)

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

myArray[0] = Date.now;  
myArray[1] = myFunction;  
myArray[2] = myCars;

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples

var x = cars.length;   // The length property returns the number of elements  
var y = cars.sort();   // The sort() method sorts arrays

Array methods are covered in the next chapters.

The length Property

The **length** property of an array returns the length of an array (the number of array elements).

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length;                       // the length of fruits is 4

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_length)

The length property is always one more than the highest array index.

Accessing the First Array Element

Example

fruits = ["Banana", "Orange", "Apple", "Mango"];  
var first = fruits[0];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_first)

Accessing the Last Array Element

Example

fruits = ["Banana", "Orange", "Apple", "Mango"];  
var last = fruits[fruits.length - 1];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_last)

Looping Array Elements

The safest way to loop through an array, is using a "for" loop:

Example

var fruits, text, fLen, i;  
  
fruits = ["Banana", "Orange", "Apple", "Mango"];  
text = "<ul>";  
  
fLen = fruits.length;  
for (i = 0; i < fLen; i++) {  
    text += "<li>" + fruits[i] + "</li>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop)

You can also use the Array.forEach() function:

Example

var fruits, text;  
  
fruits = ["Banana", "Orange", "Apple", "Mango"];  
text = "<ul>";  
  
fruits.forEach(myFunction);  
function myFunction(value) {  
    text += "<li>" + value + "</li>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop_foreach)

Adding Array Elements

The easiest way to add a new element to an array is using the push method:

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Lemon");                // adds a new element (Lemon) to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add_push)

New element can also be added to an array using the length property:

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[fruits.length] = "Lemon";     // adds a new element (Lemon) to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add)

**WARNING !**

Adding elements with high indexes can create undefined "holes" in an array:

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[6] = "Lemon";                 // adds a new element (Lemon) to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_holes)

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

Example

var person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
var x = person.length;         // person.length will return 3  
var y = person[0];             // person[0] will return "John"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_1)

**WARNING !!**  
If you use named indexes, JavaScript will redefine the array to a standard object.  
After that, some array methods and properties will produce **incorrect results**.

 Example:

var person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
var x = person.length;         // person.length will return 0  
var y = person[0];             // person[0] will return undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_2)

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays. When to use Objects.

* JavaScript does not support associative arrays.
* You should use **objects** when you want the element names to be **strings (text)**.
* You should use **arrays** when you want the element names to be **numbers**.

Avoid new Array()

There is no need to use the JavaScript's built-in array constructor **new** Array().

**Use [] instead.**

These two different statements both create a new empty array named points:

var points = new Array();         // Bad  
var points = [];                  // Good

These two different statements both create a new array containing 6 numbers:

var points = new Array(40, 100, 1, 5, 25, 10); // Bad  
var points = [40, 100, 1, 5, 25, 10];          // Good

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_literal)

The **new** keyword only complicates the code. It can also produce some unexpected results:

var points = new Array(40, 100);  // Creates an array with two elements (40 and 100)

What if I remove one of the elements?

var points = new Array(40);       // Creates an array with 40 undefined elements !!!!!

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new_error)

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator **typeof** returns "object":

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
typeof fruits;             // returns object

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_typeof)

The typeof operator returns object because a JavaScript array is an object.

Solution 1:

To solve this problem ECMAScript 5 defines a new method **Array.isArray()**:

Array.isArray(fruits);     // returns true

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray_method)

The problem with this solution is that ECMAScript 5 is **not supported in older browsers**.

Solution 2:

To solve this problem you can create your own isArray() function:

function isArray(x) {  
    return x.constructor.toString().indexOf("Array") > -1;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray)

The function above always returns true if the argument is an array.

Or more precisely: it returns true if the object prototype contains the word "Array".

Solution 3:

The **instanceof** operator returns true if an object is created by a given constructor:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
fruits instanceof Array     // returns true

20.JS Array Methods

## Converting Arrays to Strings

The JavaScript method **toString()** converts an array to a string of (comma separated) array values.

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();

### Result

Banana,Orange,Apple,Mango

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_tostring)

The **join()** method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");

### Result

Banana \* Orange \* Apple \* Mango

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_join)

## Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

## Popping

The **pop()** method removes the last element from an array:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop();              // Removes the last element ("Mango") from fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop)

The pop() method returns the value that was "popped out":

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var x = fruits.pop();      // the value of x is "Mango"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop_out)

## Pushing

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

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push)

The push() method returns the new array length:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var x = fruits.push("Kiwi");   //  the value of x is 5

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push_length)

## Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The **shift()** method removes the first array element and "shifts" all other elements to a lower index.

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();            // Removes the first element "Banana" from fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_shift)

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

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();             // Returns "Banana"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_shift_return)

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

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_unshift)

The unshift() method returns the new array length.

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon");    // Returns 5

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_unshift_return)

## Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0. [0] is the first array element, [1] is the second, [2] is the third ...

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[0] = "Kiwi";        // Changes the first element of fruits to "Kiwi"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_change)

The length property provides an easy way to append a new element to an array:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_change_add)

## Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
delete fruits[0];           // Changes the first element in fruits to **undefined**

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_delete)

Using **delete** may leave undefined holes in the array. Use pop() or shift() instead.

## Splicing an Array

The **splice()** method can be used to add new items to an array:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_splice)

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

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

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

## Using splice() to Remove Elements

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(0, 1);        // Removes the first element of fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_remove)

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

## Merging (Concatenating) Arrays

The **concat()** method creates a new array by merging (concatenating) existing arrays:

### Example (Merging Two Arrays)

var myGirls = ["Cecilie", "Lone"];  
var myBoys = ["Emil", "Tobias", "Linus"];  
var myChildren = myGirls.concat(myBoys);     // Concatenates (joins) myGirls and myBoys

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat)

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments:

### Example (Merging Three Arrays)

var arr1 = ["Cecilie", "Lone"];  
var arr2 = ["Emil", "Tobias", "Linus"];  
var arr3 = ["Robin", "Morgan"];  
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1 with arr2 and arr3

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat2)

The concat() method can also take values as arguments:

### Example (Merging an Array with Values)

var arr1 = ["Cecilie", "Lone"];  
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat3)

## Slicing an Array

The **slice()** method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

### Example

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(1);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice1)

The slice() method creates a new array. It does not remove any elements from the source array.

This example slices out a part of an array starting from array element 3 ("Apple"):

### Example

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(3);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice3)

The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

### Example

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(1, 3);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice)

If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.

### Example

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(2);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice2)

## Automatic toString()

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_tostring)

### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_automatic)

All JavaScript objects have a toString() method.