

# Javascript

PRIMITIVE TYPES

# Javascript datatypes

---

## Primitive data types:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all



## Non-primitive data types:

Data Type	Description
Object	represents instance through which we can access members

# Javascript datatypes: String

---

A string is textual content.

It must be enclosed in single or double quotation marks.

```
var str1 = "Hello World";
```

Concatenation: You can use single or double quotation marks

```
var str = 'Hello ' + "World " + 'from ' + 'TutorialsTeacher ';
```

Quotes in String: Can be applied in two ways:

```
var str1 = "This is 'simple' string";
```

```
var str1 = "This is \"simple\" string";
```

# Javascript datatypes: Code charts

---

Sec. Esc	Resultado
\\	Barra invertida
\'	Comilla simple
\"	Comillas dobles
\n	Salto de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\f	Salto de página
\r	Retorno de carro
\b	Retroceso

á	é	í	ó	ú	Á	É	Í
\u00e1	\u00e9	\u00ed	\u00f3	\u00fa	\u00c1	\u00c9	\u00cd
Ó	Ú	ü	Ü	ñ	Ñ	ï	¿
\u00d3	\u00da	\u00fc	\u00dc	\u00f1	\u00d1	\u00a1	\u00bf

<https://www.unicode.org/charts/PDF/U0080.pdf>

# Javascript datatypes: String

---

## Template literals:

- Enclosed by backtick (`) characters
- Allow multiline strings and string interpolation

```
let a = 5;  
let b = 10;  
console.log(`Fifteen is ${a + b} and  
not ${2 * a + b}.`);  
// "Fifteen is 15 and  
// not 20."
```

# Javascript datatypes: String

---

A string can also be treated like character array

Or

JavaScript also provides you String object to create a string using new keyword

## Example: String as array

```
var str = 'Hello World';

str[0] // H
str[1] // e
str[2] // l
str[3] //l
str[4] // o

str.length // 11
```

## Example: String object

```
var str1 = new String();
str1 = 'Hello World';

// or

var str2 = new String('Hello World');
```

Note: It is recommended to use primitive string instead of String object.

# Javascript datatypes: String

String methods:

([https://www.w3schools.com/jsref/jsref\\_obj\\_string.asp](https://www.w3schools.com/jsref/jsref_obj_string.asp) )

<code>charAt()</code>	Returns the character at a specified index (position)
<code>charCodeAt()</code>	Returns the Unicode of the character at a specified index
<code>concat()</code>	Returns two or more joined strings
<code>endsWith()</code>	Returns if a string ends with a specified value
<code>fromCharCode()</code>	Returns Unicode values as characters
<code>includes()</code>	Returns if a string contains a specified value
<code>indexOf()</code>	Returns the index (position) of the first occurrence of a value in a string
<code>lastIndexOf()</code>	Returns the index (position) of the last occurrence of a value in a string
<code>localeCompare()</code>	Compares two strings in the current locale
<code>match()</code>	Searches a string for a value, or a regular expression, and returns the matches
<code>repeat()</code>	Returns a new string with a number of copies of a string
<code>replace()</code>	Searches a string for a value, or a regular expression, and returns a string where the values are replaced
<code>search()</code>	Searches a string for a value, or regular expression, and returns the index (position) of the match
<code>slice()</code>	Extracts a part of a string and returns a new string
<code>split()</code>	Splits a string into an array of substrings
<code>startsWith()</code>	Checks whether a string begins with specified characters
<code>substr()</code>	Extracts a number of characters from a string, from a start index (position)
<code>substring()</code>	Extracts characters from a string, between two specified indices (positions)
<code>toLocaleLowerCase()</code>	Returns a string converted to lowercase letters, using the host's locale
<code>toLocaleUpperCase()</code>	Returns a string converted to uppercase letters, using the host's locale
<code>toLowerCase()</code>	Returns a string converted to lowercase letters
<code>toString()</code>	Returns a string or a string object as a string
<code>toUpperCase()</code>	Returns a string converted to uppercase letters
<code>trim()</code>	Returns a string with removed whitespaces
<code>valueOf()</code>	Returns the primitive value of a string or a string object

# Javascript datatypes: Number

---

Number type represents integer, float, hexadecimal, octal or exponential value.

First character in a Number type must be an integer value  
It must not be enclosed in quotation marks.

```
var int = 100;  
var float = 100.5;  
var hex = 0xffff;  
var exponential = 2.56e3;  
var octal = 030;
```



# Javascript datatypes: Number

---

JavaScript also provides Number object which can be used with **new** keyword

```
var hex = new Number(0xff);
```

The following table lists all the properties of Number type:

Property	Description
MAX_VALUE	Returns the maximum number value supported in JavaScript
MIN_VALUE	Returns the smallest number value supported in JavaScript
NEGATIVE_INFINITY	Returns negative infinity (-Infinity)
NaN	Represents a value that is not a number.
POSITIVE_INFINITY	Represents positive infinity (Infinity).

```
let a = Number.MAX_VALUE;
console.log(a + " is the maximum value");
let b = Number.MIN_VALUE;
console.log(b + " is the minimum value");
let c = (-a) * 2;
console.log(c + " is -MaxValue * 2");
let d = (a) * 2;
console.log(d + " is MaxValue * 2");
let e = c/d;
console.log(e + " is -Infinity / +Infinity");
```

# Javascript datatypes: Number

---

Number methods:  
([https://www.w3schools.com/jsref/jsref\\_obj\\_number.asp](https://www.w3schools.com/jsref/jsref_obj_number.asp))

Method	Description
<u><a href="#">isFinite()</a></u>	Checks whether a value is a finite number
<u><a href="#">isInteger()</a></u>	Checks whether a value is an integer
<u><a href="#">isNaN()</a></u>	Checks whether a value is Number.NaN
<u><a href="#">isSafeInteger()</a></u>	Checks whether a value is a safe integer
<u><a href="#">toExponential(x)</a></u>	Converts a number into an exponential notation
<u><a href="#">toFixed(x)</a></u>	Formats a number with x numbers of digits after the decimal point
<u><a href="#">toLocaleString()</a></u>	Converts a number into a string, based on the locale settings
<u><a href="#">toPrecision(x)</a></u>	Formats a number to x length
<u><a href="#">toString()</a></u>	Converts a number to a string
<u><a href="#">valueOf()</a></u>	Returns the primitive value of a number

# Javascript datatypes: Boolean

---

Boolean is a primitive data type in JavaScript. Boolean can have only two values, true or false.

```
var YES = true;  
var NO = false;
```

JavaScript includes Boolean object to represent true or false. It can be initialized using **new** keyword.

```
var bool = new Boolean(true);  
alert(bool); // true
```

# Javascript datatypes: Boolean

---

JavaScript treats empty string (""), 0, undefined, NaN and null as false. Everything else is true.

```
var bool1 = new Boolean(""); // false
var bool2 = new Boolean(0); // false
var bool3 = new Boolean(undefined); // false
var bool4 = new Boolean(null); // false
var bool5 = new Boolean(NaN); // false
var bool6 = new Boolean("some text"); // true
var bool7 = new Boolean(1); // true
```

# Javascript datatypes: Null and undefined

---

A null means absence of a value. It means we have defined a variable but have not assigned any value yet, so value is absence.

```
var myVar = null;  
  
alert(myVar); // null
```

A variable or an object has an undefined value when no value is assigned before using it. Undefined means lack of value or unknown value.

```
var myVar;  
  
alert(myVar); // undefined
```

# JavaScript Arithmetic Operators

---

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

```
var x = 5;  
var y = 2;  
var z = x % y; //1 --Division remainder
```

```
var a, b;  
a = 3;  
b = 5;  
alert (a);  
alert (b);  
a = ++b;  
alert (a);  
alert (b);
```

```
a=3;  
b=5;  
alert (a);  
alert (b);  
a = b++;  
alert (a);  
alert (b);
```

# JavaScript Assignment Operators

---

Operator	Example	Same As
=	<code>x = y</code>	<code>x = y</code>
+=	<code>x += y</code>	<code>x = x + y</code>
-=	<code>x -= y</code>	<code>x = x - y</code>
*=	<code>x *= y</code>	<code>x = x * y</code>
/=	<code>x /= y</code>	<code>x = x / y</code>
%=	<code>x %= y</code>	<code>x = x % y</code>

# 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

Ternary operator:

Syntax

```
variablename = (condition) ? value1:value2
```

Example

```
var voteable = (age < 18) ? "Too young":"Old enough";
```



# JavaScript Comparison Operators: Example

*Use three equals unless you fully understand the conversions that take place for two-equals:*

Given that **x = 5**:

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true

# JavaScript Comparison Operators: Example

Case	Value
2 < 12	true
2 < "12"	true
2 < "John"	false
2 > "John"	false
2 == "John"	false
"2" < "12"	false
"2" > "12"	true
"2" == "12"	false

To secure a proper result, variables should be converted to the proper type before comparison:

```
age = Number(age);  
if (isNaN(age)) {  
    voteable = "Error in input";  
} else {  
    voteable = (age < 18) ? "Too young" : "Old enough";  
}
```

# JavaScript Logical Operators

---

Operator	Description
&&	logical and
	logical or
!	logical not

Example: Given that **x = 6** and **y = 3**

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

# 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

```
typeof "John"           // Returns "string"
typeof 3.14              // Returns "number"
typeof NaN              // Returns "number"
typeof false            // Returns "boolean"
typeof [1,2,3,4]         // Returns "object"
typeof {name:'John', age:34} // Returns "object"
typeof new Date()        // Returns "object"
typeof function () {}    // Returns "function"
typeof myCar             // Returns "undefined"
typeof null             // Returns "object"
```

# JavaScript Bitwise Operators

---

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

# JavaScript operator precedence values

Level	Operators	Notes
1	() [] .	call, member (including typeof and void)
2	! ~ - ++ --	negation, increment
3	* / %	multiply/divide
4	+ -	addition/subtraction
5	<< >> >>>	bitwise shift
6	< <= > >=	relational
7	== !=	equality
8	&	bitwise AND
9	^	bitwise XOR
10		bitwise OR
11	&&	logical AND
12		logical OR
13	?:	conditional
14	= += -= *= /= %<= <<= >>= >>>= &= ^=  =	assignment

# Type conversion

---

JavaScript variables can be converted to a new variable and another data type:

- By the use of a JavaScript function (Explicit)
- **Automatically** by JavaScript itself (Implicit)

Automatic type conversion happens in many cases. In most cases it is somewhat logical, in others you either need to know exactly what you're doing or you need to do an explicit type conversion.

# Implicit Type conversion

---

## Automatic conversion using arithmetic operators:

**+** operator: If any of the operands of the addition is a string, Javascript will try to convert the other operand into a string too.

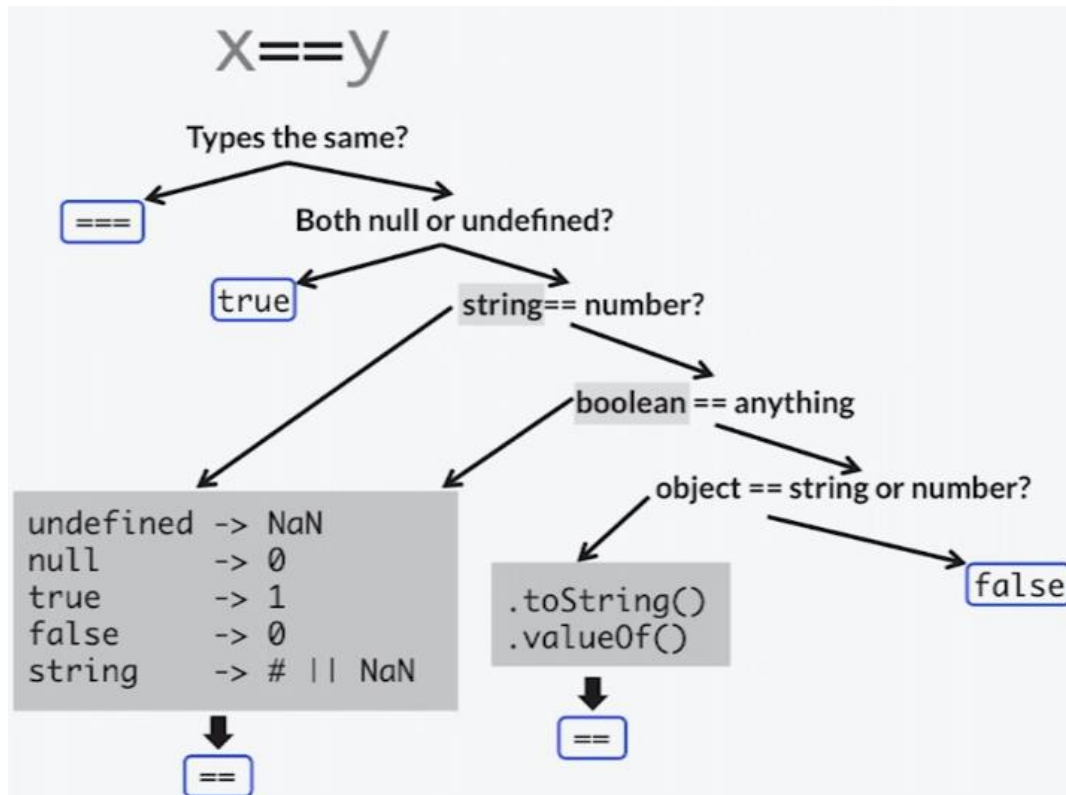
Any other arithmetic operator: If any of the operands of the operation is a string, Javascript will try to convert it into a number. If this is not possible, it would take value NaN.

<code>5 + null</code>	<code>// returns 5</code>	because null is converted to 0
<code>"5" + null</code>	<code>// returns "5null"</code>	because null is converted to "null"
<code>"5" + 2</code>	<code>// returns 52</code>	because 2 is converted to "2"
<code>"5" - 2</code>	<code>// returns 3</code>	because "5" is converted to 5
<code>"5" * "2"</code>	<code>// returns 10</code>	because "5" and "2" are converted to 5 and 2



# Implicit Type conversion

## Automatic conversion with operator ===:



- true gets value 1 within a == comparison
- false gets value 0 within a == comparison
- When comparing an string with a number, Javascript tries to convert the string into a decimal or an hexadecimal number. If this is not posible, it would take value NaN except for the empty string, that becomes a 0.

# Implicit Type conversión

---

Some interesting cases...

```
' ' == '0'  
0 == ''  
0 == '0'  
  
false == 'false'  
false == '0'  
  
false == undefined  
false == null  
null == undefined  
  
' \t\r\n ' == 0
```

```
"0" == true  
"0.1e1" == true
```

# Explicit type conversion

---

The global method **String()** convert different objects to strings:

```
String(x)          // returns a string from a number variable x
String(123)         // returns a string from a number literal 123
String(100 + 23)    // returns a string from a number from an expression
String(false)       // returns "false"
String(true)        // returns "true"
String(Date())       // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"
```

The `toString()` method returns the value of a String object.

```
x.toString()
(123).toString()
(100 + 23).toString()
false.toString()    // returns "false"
true.toString()     // returns "true"
Date().toString()

(15).toString(2)    // returns "1111" → Converts to binary value
(15).toString(16)   // returns "f" → Converts to hexadecimal value
```

# Explicit type conversion

---

The global method **Number()** can convert different objects to numbers

```
Number("3.14")    // returns 3.14
Number(" ")       // returns 0
Number("")         // returns 0
Number("99 88")   // returns NaN
Number(false)     // returns 0
Number(true)      // returns 1
d = new Date();
Number(d)          // returns 1404568027739

Number("2")+2     // returns 22
Number(`2`)+2     // returns 4
```

# Explicit type conversion

---

The global method **Boolean()** can convert different objects to booleans

Everything With a "Real" Value is True

```
var b1 = Boolean(100); //true
var b2 = Boolean(3.14); //true
var b3 = Boolean(-15); //true
var b4 = Boolean("Hello"); //true
var b5 = Boolean('false'); //true
var b6 = Boolean(1 + 7 + 3.14); //true
```

Everything Without a "Real" Value is False

```
var b1 = Boolean(0); //false
var b2 = Boolean(-0); //false
var b3 = Boolean(''); //false
var b4 = Boolean(undefined); //false
var b5 = Boolean(null); //false
var b6 = Boolean(false); //false
var b7 = Boolean(10 / "H"); //false --> Es NaN
```

# Explicit type conversion

---

The `toString()` method returns the value of a String object.

Method	Description
<code>toExponential()</code>	Returns a string, with a number rounded and written using exponential notation.
<code>toFixed()</code>	Returns a string, with a number rounded and written with a specified number of decimals.
<code>toPrecision()</code>	Returns a string, with a number written with a specified length
<code>parseFloat()</code>	Parses a string and returns a floating point number
<code>parseInt()</code>	Parses a string and returns an integer

**Examples:**

```
parseInt("38.5kilos") // returns 38
```

```
parseFloat('38.5kilos') // returns 38.5
```