Typeof in JavaScript is an operator used for type checking and returns the data type of the operand passed to it. The operand can be any variable, function, or object whose type you want to find out using the typeof operator.

JavaScript's number feature enables you to represent numeric and floating numbers. JavaScript number has a particular value called NaN function, which means for Not-a-Number.  It is a property of the global object, and it is used to represent a value that cannot be represented as a number. NaN can be the result of an operation that was supposed to return a number, but couldn't because of an error or undefined/empty value. For example, dividing zero by zero or attempting to parse a string that doesn't contain a valid number will result in NaN.

*// NaN is a number that is not a legal number.*  
console.log(NaN); *// NaN*  
  
*// NaN is the result of an operation that was supposed to return a number, but couldn't because of an error or undefined/empty value.*  
console.log(1 / 0); *// NaN*  
console.log(parseInt("abc")); *// NaN*  
  
*// NaN is not equal to any other value, including itself.*  
console.log(NaN === NaN); *// false*  
  
*// NaN is not greater than, less than, or equal to any other value.*  
console.log(NaN > 1); *// false*  
console.log(NaN < 1); *// false*  
console.log(NaN == 1); *// false*  
  
*// NaN is the only value that is not equal to itself.*  
console.log(NaN !== NaN); *// true*

In JavaScript NaN is short for "Not-a-Number". The isNaN() method returns true if a value is NaN. The isNaN() method converts the value to a number before testing it.

The isNaN() function in JavaScript is used to check if a value is NaN (Not a Number). It returns true if the value is NaN, and false if it is a number or can be converted to a number.

NaN is a special value in JavaScript that represents an undefined or unrepresentable value. It can be the result of an invalid mathematical operation, such as dividing by zero, or trying to parse a string that is not a number.

var x = "abc";  
  
if (isNaN(x)) {  
 console.log("x is not a number");  
} else {  
 console.log("x is a number");  
}

In JavaScript, string interpolation is a process of inserting or placing an expression inside of a string. This can be done using template literals, which are strings wrapped in backticks (` `). Inside the template literal, you can use the ${expression} placeholder to insert the value of an expression. The expression can be a variable, a function call, or even a mathematical expression. In string concatenation, it is hard to maintain strings as they grow large it becomes tedious and complex. In order to make it readable, the developer has to maintain all the whitespaces. This is where ES6 comes to the rescue with String interpolation. In JavaScript, the template literals (strings wrapped in backticks ` `) and ${expression} as placeholders perform the string interpolation. Now we can write the above myInfo function with string interpolation.

String Interpolation

function myInfo(fname, lname, country) {

    return `My name is ${fname} ${lname}. ${country} is my favorite country`;

}

console.log(myInfo("john", "doe", "India"));

Concatenation in JavaScript is the process of joining two or more strings together. This can be done using the + operator or the concat() method.

function myInfo(fname, lname, country) {

    return "My name is " + fname + " " + lname + ". "

            + country + " is my favorite country.";

}

console.log(myInfo("john", "doe", "India"));

Wrapping JavaScript functions lets you add common logic to functions you do not control, like native and external functions. Many JavaScript libraries, like the TrackJS agents, need to wrap external functions to do their work.

function wrap(oldFunction) {

// return a new function that will call the oldFunction

// with all of the arguments passed to it

return (...args) => {

// log the arguments passed to the wrapped function

console.log(args);

// call the old function with all of the arguments

return oldFunction(...args);

}

}

// create the newly wrapped add function

const newAdd = wrap(add);

const sum = newAdd(1, 3); // [1, 3]

console.log(sum); // 4