JavaScript Notes

Draft

# JavaScript History

<https://www.w3schools.com/js/js_history.asp>

# Variables, types and values

## Primitive Data types

In JavaScript, a primitive (primitive value, primitive data type) is data that is not an object and has no properties or methods. Primitive values are immutable (they are hardcoded and cannot be changed).

There are 7 primitive data types:

|  |  |
| --- | --- |
| Type | Description |
| boolean | Boolean represents a logical entity and can have two values: true and false. |
| number | The Number type is a double-precision 64-bit value. It is capable of storing floating-point numbers between 2^-1074 and 2^1024, but can only safely store integers in the range -(2^53 − 1) to 2^53 – 1. |
| bigint | The largest number that JavaScript can reliably represent with the Number primitive is 253-1. Bigint provides a way to represent whole numbers larger than 253-1 |
| string | It is used to represent textual data. It is a set of "elements" of 16-bit unsigned integer values. Each element in the String occupies a position in the String. The first element is at index 0, the next at index 1, and so on. The length of a String is the number of elements in it. |
| undefined | A variable that has not been assigned a value has the value undefined. |
| null | The null type has exactly one value: null. A null value represents a reference that points, generally intentionally, to a nonexistent or invalid object or address. |
| symbol | A Symbol is a unique and immutable primitive value and may be used as the key of an Object property |

Except for null and undefined, all primitive values have object equivalents that wrap around the primitive values

Primitive values are immutable (they are hardcoded and cannot be changed). If a = 10, you can change the value of a, but you cannot change the value of 10.

## Non-Primitive Data Types

|  |  |
| --- | --- |
| Type | Description |
| Object | An object is a value in memory which is possibly referenced by an identifier. In JavaScript, objects can be seen as a collection of properties. |
| Array | Array enables storing a collection of multiple items under a single variable name, and has members for performing common array operations. |
| RegExp | Represents regular expression |

## How to declare variables

 In JavaScript, a variable stores the data value that can be changed later on.

There are three ways we can declare a variable.

1. var
2. let
3. const

In following section, we will use var to declare variables. In the later sections we will review in detail the difference and usage of above mentioned three ways of variable declaration.

Example, to declare a variable:

var num = 10; // declares a variable of type number

To display something on browser’s console, we can use

window.console.log(“Welcome”);

or simply,

console.log(“Welcome”);

The Window.console property returns a reference to the console object, which provides methods for logging information to the browser's console. We will discuss Window in DOM section, later.

JavaScript is loosely typed. You don’t have to tell that a string is a string, nor you can require a function to accept an integer as its parameter.

var num = 5

console.log(num) // output 5

num = "Hello"

console.log(num) // output Hello

## Number

A number data type can be an integer, a floating point value, an exponential value, a ‘NaN’ or a ‘Infinity’.

var a=10;  // integer value

var b=10.5;  // a number containing a decimal

var c = 10e2 //  an exponential value which evaluates to 10\*100;

If a number is divided by 0, the result is infinity.

10/0 //infiniry

The type of infinity is a number

typeof(infinity);   // returns number

When we perform an operation on a number we get NaN (not a number) value

“Hello” \* 10; // retunrs NaN

console.log(10); // will display 10

vaar num = 20;

console.log(num) // will display value stored in num. i.e., 20

# Operators

## Arithmetic operators

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

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

## Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Comparing** | **Returns** |
| **==** | 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 |
| **>** | greater than | x > 8 | FALSE |
| **<** | less than | x < 8 | TRUE |
| **>=** | greater than or equal to | x >= 8 | FALSE |
| **<=** | less than or equal to | x <= 8 | TRUE |

## Logical operators

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

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

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

## Conditional (Ternary) Operator

JavaScript has a conditional operator that assigns a value to a variable based on some condition.

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

## Bitwise operators

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shifts left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |
| >>> | Zero fill right shift | Shifts right by pushing zeros in from the left, and let the rightmost bits fall off |

## Examples

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Result | Same as | Result |
| 5 & 1 | 1 | 0101 & 0001 | 0001 |
| 5 | 1 | 5 | 0101 | 0001 | 0101 |
| ~ 5 | 10 | ~0101 | 1010 |
| 5 << 1 | 10 | 0101 << 1 | 1010 |
| 5 ^ 1 | 4 | 0101 ^ 0001 | 0100 |
| 5 >> 1 | 2 | 0101 >> 1 | 0010 |
| 5 >>> 1 | 2 | 0101 >>> 1 | 0010 |

# String

It is used to represent textual data. Each element in the String occupies a position in the String. The first element is at index 0, the next at index 1, and so on. The length of a String is the number of elements in it. String is zero or more characters written inside quotes

“Hello world” // string with double quotes

‘Hello world’ // string with single quotes

var str = “Hello world” // string assigned to variable x

## Strings and arithmetic operations

With string + is used for concatenation. For example, ‘Hello ’ + ‘ world’; results in ‘Hello world’.

We can say String + String = String.

var str1 = “Hello “

var str 2 = “world”

console.log (str1 + str2) // Hello world

But what will happen if we have different data types, for example, number?

In JavaScript, if it finds a string with + it converts the other parameter to string.

String + Number = String

’12' + 12; // '1212'

13 + '1'; // '131'

12.34 + '34'; // "12.3434"

But what about other operations?

'12' - 10; // 2

'90.10' - 10; // 80.10

'12' \* 12; // 144

'12' \* '12'; // 144

'12' / '12'; // 1

We can say, String (- \* /) String = Number &

String (- \* /) Number = Number

Provided operands are valid numbers.

'we' - 2; // NaN

':(' \* 2; // NaN

'&9' / 2; // NaN

// We have seen this

'10' + 10; // '1010'

// But

+'12' + 10; // 22

Here the + before '12' is a unary operator which precedes its operand and evaluates to its operand but attempts to convert it into a number, if it isn't already.

+'10'; // 10

+''; // 0

Guess the output

10 + '0' - 30;

## String Methods

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

JavaScript has object equivalents that wrap around the String primitive value. This provides useful methods for string handling.

### Length of String

str.length

var str = “Hello wrold’;

console.log(str.length); //11

### Returning nth character of String

str[n]

var str="Hello"

console.log(str[0]) //H

### trim() – Removing white space

var str = “ Hello”;

console.log(str.trim());

### ****includes() – Check if string contains substring:****

var str = “Lahore City”;

var str2 = 'City';

console.log(str.includes(str2));

//true

### **indexOf() — Find the index of a substring**

var str =”He is my friend”;

var str2 = 'my';

var str3 = 'your';

console.log(str.indexOf(str2));

//6

console.log(str.indexOf(str3));

//-1

### ****toUpperCase() — Capitalizes entire string****

var str = “Hello”;

console.log(str.toUpperCase());

//HELLO

### ****.toLowerCase() — Lower cases entire string****

var str = 'Hello';

console.log(str.toLowerCase());

//hello

### ****replace() — Replaces strings with new values****

var str = “Aslam Khan”;

var str1 = “Aslam”;

var str2 = “Ali”

console.log(str.replace(str1, str2));

//Ali Khan

### ****slice() — Return a section of a string****

Extract a section of string str

// from position 2 to end

var str = “karachi;

console.log(str.slice(2));

//rachi

// from 0 to 3, but 3 not included

var str1 = "karachi";

var str2 = str1.slice(0, 3);

console.log(str2);

//kar

### ****.split() — Converts string into an array of strings****

var str = “Karachi,Pakistan”;

console.log(str.split(","));

//["karachi", "Pakistan”]

### ****.****repeat() — Repeats a string a specified number of times

var str = “Pakistan”;

console.log(str.repeat(2));

//pakistanpakistan

### ****.match() — Returns array of matching strings:****

var str = “He is my friend”;

var str1 = “my”;

console.log(str.match(str1));

//["my"]

### ****.charAt() — Returns the character at an index:****

var str = '”Pakistan”;

console.log(str.charAt(2));

//a

### ****.concat() – Adding the two strings:****

var str = “Lahore “;

var str1 = “Pakistan”;

console.log(str.concat(str1));

//”Lahore Pakistan”

### ****.search() – Find the string:****

var str = “Lahore Pakistan;

var str1 =”Lahore”;

console.log(str.search(str1));

//0 means true, otherwise will return -1

### ****substring() – Extracts the characters from a string****

var str = 'pakistan ';

console.log(str.substring(2));

//kistan

From element 1 to 3-1 (excluding end)

var str = 'pakistan ';

console.log(str.substring(1,3));

//ak

### ****.substr()- Extracts parts of a string****

Extract 4 characters starting from element 2

var str = “pakistan;

console.log(str.substr(2, 4));

//kist

### .valueOf() – Return the primitive value of the string

var str = “Pakistan”;

console.log(str.valueOf());

//pakistan

### ****.startsWith() – Check if a string starts with****

var str = “you are my friend”;

console.log(str.startsWith('you’));

//true

### ****.endsWith() – Check if a sting ends with****

var str = “you are my friend”;

console.log(str.endsWith('friend'));

//true

### ****.lastIndexOf() – returns the position of the last occurrence of a specified value in a string****

var str = "This is apple, apple is sweet"

console.log(str.lastIndexOf('is'));

//21

### .indexOf() - the first occurrence of a value in a string

var str = "This is apple, apple is sweet"

console.log(str.indexOf('is'));

//2

## Converting strings to integers and decimals

If you want to do addition, you must convert any strings to numbers.

parseInt converts all strings to integers.

var x = ’10’

var y = parseInt(x) + 1;

It doesn't round. It simply lops off the decimals. In the following statement, num is assigned not 2 as you might expect, but 1.

var num = parseInt("1.9999");

To preserve any decimal values, use parseFloat. In the following statement myFractional is assigned 1.9999.

var myFractional = parseFloat("1.9999");

## Converting strings to numbers, numbers to strings

You can deal the distinction between integers and floatingpoint numbers by using Number. This handy conversion tool converts a string representing either an integer or a floating-point number to a number that's identical to the one inside the parentheses. The following code converts the string "24" to the number 24.

1 var integerString = "24"

2 var num = Number(integerString);

The following code converts the string "24.9876" to the number 24.9876.

1 var floatingNumString = "24.9876";

2 var num = Number(floatingNumString);

## Converting strings to numbers, numbers to strings

Converting a number to a string, perhaps so you can format it, is straightforward.

var numberAsNumber = 1234;

var numberAsString = numberAsNumber.toString();

The code above converts the number 1234 to the string "1234" and assigns it to the variable numberAsString

# Array

Let's assign some string values to variables.

var city0 = "Islamabad";

var city1 = "Karachi";

var city2 = "Lahore";

var city3 = "Peshawer";

var city4 = "Quetta";

The variable names are all the same, except they end in different numbers.

Now, having made these assignments, if I code...

console.log("Welcome to " + city3);

...a message will displays saying, "Welcome to Peshawer".

What, if our list of cities is growing or we have to access all these cities somewhere in the code……. I would be nightmare. Too many variables, calling, accessing……..

The solution to handle such data is Array.

The Array object, enables storing a collection of multiple items under a single variable name, and has members for performing common array operations.

JavaScript array has the following characteristics:

1. First, an array can hold values of mixed types. For example, you can have an array that stores elements with the types number, string, and boolean.
2. Second, the size of an array is dynamic and auto-growing. In other words, you don’t need to specify the array size upfront.

To create an empty array, you use square brackets without specifying any element like this

var arr = [];

The following example creates the array that holds city names, string elements:

var arr = [“Islamabad”,”Karachi”,”Lahore”]

The following example creates the array that holds number elements:

var arr = [1,2,3,4]

You can even mix the different types in the same array

var mixedArray = [1, "Ali", "Now is", true];

## Accessing JavaScript array elements

JavaScript arrays are zero-based indexed. In other words, the first element of an array starts at index 0, the second element starts at index 1, and so on.

To access an element in an array, you specify an index in the square brackets []

var arr = [“Islamabad”,”Karachi”,”Lahore”]

console.log(arr[0]); // Islamabad

console.log(arr[1]); // Karachi

console.log(arr[2]); // Lahore

To change the value of an element, you assign that value to the element like this:

arr[2]=”Faisalabad”

console.log(arr[2]); // Faisalabad

## Array Methods

### .toString() - converts an array to a string

The JavaScript method toString() converts an array to a string of (comma separated) array values.

var fruits = ["Banana", "Orange", "Apple", "Mango"];

var tstr = fruits.toString();

console.log("These are " + tstr);

### .length() - Length of Array

var arr = [“Islamabad”,”Karachi”,”Lahore”]

console.log(arr.length); // 3

### .push() - Adding element(s) to the end of an array

Using push(), you can add one or more elements to the end of an array. The following code adds two new elements to the end of the array. The push() method returns the new array’s length.

var arr = [“Islamabad”,”Karachi”,”Lahore”]

arr.push(“Faisalabad”, “Multan”);

console.log(arr);

You can also add an element to existing array.

var arr = [“Islamabad”,”Karachi”,”Lahore”]

arr[3] = “Faisalabad”

console.log(arr)

// ['Islamabad', 'Karachi', 'Lahore', 'faisalabad']

If you skip any element, it would be empty/undefined. In following example we skip arr[3]

var arr = [“Islamabad”,”Karachi”,”Lahore”]

arr[4] = “Faisalabad”

console.log(arr)

// ['Islamabad', 'Karachi', 'Lahore', empty, 'faisalabad']

### .pop() - removes the last element

The pop() method removes the last element from an array and returns that element. The length of the array is also changed.

The pop() method removes the last element from an array:

var arr = [“Islamabad”,”Karachi”,”Lahore”]

arr.pop();

The pop() method returns the value that was "popped out“.

### .shift() – removes first element

The shift() method removes the first array element and "shifts" all other elements to a lower index. In this case “Islamabad” will be removed from array.

var arr = [“Islamabad”,”Karachi”,”Lahore”]

arr.shift();

The shift() method returns the value that was “shift out“.

### .unshift() – add elements to the beginning of array

To add one or more elements to the beginning of an array, use the unshift method. The following code adds two elements to the beginning of the array.

var arr = [“Islamabad”,”Karachi”,”Lahore”]

arr.unshift(“Multan”,”Faisalabad”);

The unshift() method returns the new array length.

### .splice() - insert one or more elements

Use the splice method to insert one or more elements anywhere in an array, while optionally removing one or more elements that come after it. .

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.splice(2, 0, "Lemon", "Kiwi");

Result: Banana,Orange,Lemon,Kiwi,Apple,Mango

* 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.
* The splice() method returns an array with the deleted items.

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.splice(2, 2, "Lemon", "Kiwi");

Result: Banana,Orange,Lemon,Kiwi

Removed Items: Apple,Mango

### .slice() – slice out piece of 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"):

var fruits = ["Banana", "Orange", "Apple", "Mango"];

var citrus = fruits.slice(1);

The slice() method creates a new array.

The slice() method does not remove any elements from the source array.

fruits.slice(1, 3)

The method then selects elements from the start argument, and up to (but not including) the end argument.

.indexOf() –

The indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

var fruits = ["Banana", "Orange", "Apple", "Mango", “Banana”];

fruits.indexOf(“Banana”) //0

fruits.indexOf(“Banana”,1) //4

# For Loop

for (var i = 0; i <= 4; i++) {

console.log(i);

}