JAVASCRIPT TUTORIAL

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

**The <script> Tag**

In HTML, JavaScript code is inserted between <script> and </script> tags. Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both. Placing scripts at the **bottom** of the <body> element improves the display speed, because script interpretation slows down the display.

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

**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.

**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>

* **External scripts cannot contain <script> tags.**

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

Example

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

**JavaScript Display Possibilities**

JavaScript can "display" data in different ways:

Writing into an HTML element, using **innerHTML**. : document.getElementById("demo").innerHTML = 5 + 6;

Writing into the HTML output using **document.write()**. document.write() after an HTML document is loaded, will **delete all existing HTML**:

Writing into an alert box, using **window.alert()**.

Writing into the browser console, using **console.log()**.

**Printing Page :** to print the current page from printer.

<button onclick="window.print()">Print this page</button>

**DECLARING A VARIABLE**

let x,y,z; declaring 3 variables.

There are 3 ways to declare a JavaScript variable:

* Using **var**
* Using **let**
* Using **const**

**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;

Variables defined with var are **hoisted** to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

carName = "Volvo"; var carName;

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

**LET :->**

Variables defined with let cannot be Redeclared.

Variables defined with let must be Declared before use.

Variables defined with let have Block Scope.

Using a let variable before it is declared will result in a ReferenceError:

Example

carName = "Saab";  
let carName = "Volvo"; -🡪 referenceError

**CONST**

Variables defined with const cannot be Redeclared.

Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope.

* JavaScript const variables must be assigned a value when they are declared:
* Using a const variable before it is declared will result in a ReferenceError:

**But CONST CAN:**

* **Change the elements of constant array**
* **Change the properties of constant object**
* JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

**KEYWORDS**

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| var | Declares a variable |
| let | Declares a block variable |
| const | Declares a block constant |
| if | Marks a block of statements to be executed on a condition |
| switch | Marks a block of statements to be executed in different cases |
| for | Marks a block of statements to be executed in a loop |
| function | Declares a function |
| return | Exits a function |
| try | Implements error handling to a block of statements |

**COMMENTS**

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

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

**TYPE OPERATOR**

typeof 🡪 Returns the type of a variable

instanceof 🡪 Returns true if an object is an instance of an object type

var a = 2

console.log(typeof(a)) // number

var name = "satish"

console.log(typeof(name)) // string

**ARTHIMETIC OPERATOR**

\*\* 🡪 Exponentation

// exponenation

var x = 2;

var y= 3;

x \*\*= y;

console.log(x) // 2^3

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

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

**JS OBJECTS**

JavaScript objects are written with curly braces {}. Object properties are written as name:value pairs, separated by commas.

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

**Accessing Object Properties**

You can access object properties in two ways:

*objectName.propertyName* or *objectName["propertyName"]*

Objects can also have **methods**.

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

Comparing two JavaScript objects **always** returns **false**.

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

**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:

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

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

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

**JS FUNCTIONS**

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

**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  
}

**JS EVENTS**

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events. JavaScript lets you execute code when events are detected.

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

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>

**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 |

**STRING**

Length of string : varName.length

Do not create Strings objects.

The new keyword complicates the code and slows down execution speed.

String objects can produce unexpected results:

* **JS Objects can not be compared and so the result is always false.**

let x = new String("John");  
let y = new String("John");

console.log(x==y) // gives false as both x and y are objects and can’t be compared.

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

**STRING METHODS**

Extracting String Parts

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

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

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

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

**The String substring() Method**

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

**substr() is similar to slice().**

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

**Replacing String Content**

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

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

**Converting to Upper and Lower Case**

A string is converted to upper case with **toUpperCase**() and lower using **toLowerCase**():

let text1 = "Hello World!";  
let text2 = text1.toUpperCase();

let text3 = text1.toLowerCase(); // to lower case

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

trim() : to remove blank spaces

charAt(n) : returns character at specified index

charCodeAt(n) : returns unicode of the character at a specified index in a string

**Converting a String to an Array**

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

Example

text.split(",")    // Split on commas  
text.split(" ")    // Split on spaces  
text.split("|")    // Split on pipe

**JavaScript methods for searching strings:**

* String indexOf()
* String lastIndexOf()
* String startsWith()
* String endsWith()

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

**String search()**

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

**String match()**

The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.

**String includes()**

The includes() method returns true if a string contains a specified value.

**Quotes Inside Strings**

With **template literals**, you can use both single and double quotes inside a string:

Example : let text = `He's often called "Johnny"`;

Template literals provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is: ${...}

let firstName = "John";  
let lastName = "Doe";  
let text = `Welcome ${firstName}, ${lastName}!`;

Automatic replacing of variables with real values is called **string interpolation**.

**JS NUMBERS**

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

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

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

**Numeric Strings** : JavaScript strings can have numeric content:

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

This will work:

let x = "100";  
let y = "10";  
let z = x / y; // 10

let z = x \* y; // 1000

let z = x - y; // 90

let z = x + y; // 10010 // concatenate happens here

**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):

let x = 100 / "Apple"; // NaN

You can use the global JavaScript function isNaN() to find out if a value is a not a number:

NaN is a number: typeof NaN returns number:

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

Division by 0 (zero) also generates Infinity:

Example

let x =  2 / 0; // Infinity  
let y = -2 / 0; // Infinity

Infinity is a number: typeof Infinity returns number.

**NUMBER METHODS**

toString() method returns a number as a string.

X = 123

x.toString()

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

let x = 9.656;  
x.toExponential(2); 9.66e+0  
x.toExponential(4); 9.6560e+0

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

let x = 9.656;  
x.toFixed(0); // 10  
x.toFixed(2); // 9.66  
x.toFixed(4); // 9.6560

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

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

console.log("=================== NUMBER METHOD ============")

console.log(Number(true)); // 1

console.log(Number(false)); //0

console.log(Number("10")); // 10

console.log(Number("  10")); // 10

console.log(Number("10  ")); //10

console.log(Number("10.33")); // 10.33

console.log(Number("10,33")); // NaN

console.log(Number("10 33")); // NaN

console.log(Number("John")); //NaN

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

console.log(parseInt("-10")); // -10

console.log(parseInt("-10.33")); // -10

console.log(parseInt("10")); // 10

console.log(parseInt("10.33")); //10

console.log(parseInt("10 20 30"));// 10

console.log(parseInt("10 years"));//10

console.log(parseInt("years 10")); //NaN

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

console.log(parseFloat("10")); // 10

console.log(parseFloat("10.33")); //10.33

console.log(parseFloat("10 20 30"));// 10

console.log(parseFloat("10 years")); //10

console.log(parseFloat("years 10")); //NaN

**JS ARRAYS**

it is a common practice to declare arrays with the **const** keyword.

const cars = ["Saab", "Volvo", "BMW"];

You can also create an array, and then provide the elements:

Example

const cars = [];  
cars[0]= "Saab";  
cars[1]= "Volvo";  
cars[2]= "BMW";

creating array with new keyword : const cars = new Array("Saab", "Volvo", "BMW");

**Access the Full Array** :->With JavaScript, the full array can be accessed by referring to the array name:

Example

const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;

typeof(cars) // objects

Arrays are a special type of objects.  Arrays use **numbers** to access its "elements". person[0] returns 0th item. Whereas Objects use **names** to access its "members". person.firstName returns firstName propery of object person.

**Printing array elements on WebPage:**

<!-- ACCESSING ARRAY ELEMENTS -->

<br>

<h3> Priniting Array Elements </h3>

<p id="my\_array"></p>

<script>

    const my\_cars = ['bmw', 'volovl', 'mercedes']

    text = "<ul>"

    for (let i=0;i<my\_cars.length;i++){

        text += "<li>" + my\_cars[i] + "</li>";

    }

    text += "</ul>"

    document.getElementById("my\_array").innerHTML = text;

</script>

**USING FOR EACH FUNCTION**

**<!-- using for each function -->**

**<p id="fruits\_place"></p>**

**<script>**

**const fruits = ["apple", "banana", "Guava"]**

**text = "<ul>"**

**fruits.forEach(my\_function)**

**text+= "</ul>"**

**function my\_function(fruit){**

**text+= "<li>" + fruit +"</li>";**

**}**

**document.getElementById("fruits\_place").innerHTML = text;**

**</script>**

**forEach()** method calls a function (a callback function) once for each array element.

**Adding Array Elements**

The easiest way to add a new element to an array is using the **push()** method:

Example

const fruits = ["Banana", "Orange", "Apple"];  
fruits.push("Lemon");  // Adds a new element (Lemon) to fruits

New element can also be added to an array using the length property:

Example

const fruits = ["Banana", "Orange", "Apple"];  
fruits[fruits.length] = "Lemon";  // Adds "Lemon" to fruits

Adding elements with high indexes can create undefined "holes" in an array: HOLES MEANS VACANT INDEXES(UNDEFINED)

**IMPORTANT**

const points = [40]; is not the same as: const points = new Array(40);

const points = [40]; // Create an array with one element:

const points = new Array(40);  // Create an array with 40 undefined elements:  
to check if variable is array or not : Array.isArray(fruits);

**JS ARRAY METHODS**

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

**join()** method also joins all array elements into a string. It behaves just like toString(), but in addition you can specify the separator:

**pop()** method removes the last element from an array: returns the value that was "popped out":

**push()** method adds a new element to an array (at the end): returns the new array length:

**shift()** method removes the first array element and "shifts" all other elements to a lower index.  returns the value that was "shifted out":

**unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements: unshift() method returns the new array length.

**length** property provides an easy way to append a new element to an array:

Array elements can be **deleted** using the JavaScript operator delete. Using delete leaves undefined holes in the array.

Use pop() or shift() instead.

**concat()** method creates a new array by merging (concatenating) existing arrays:

const my\_merged\_array = arr1.concat(arr2);

The concat() method can take any number of array arguments: const myChildren = arr1.concat(arr2, arr3);

**splice()** method can be used to add new items to an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");

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:

**slice()** method slices out a piece of an array into a new array. The slice() method creates a new array. The slice() method does not remove any elements from the source array.

**sort**() method sorts an array alphabetically:

**reverse**() method reverses the elements in an array.

**Find the Highest (or Lowest) Array Value**

There are no built-in functions for finding the max or min value in an array.

However, after you have sorted an array, you can use the index to obtain the highest and lowest values.

Sorting ascending:

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});  
// now points[0] contains the lowest value  
// and points[points.length-1] contains the highest value

**JavaScript Array map()**

The map() method creates a new array by performing a function on each array element.

The map() method does not execute the function for array elements without values.

The map() method does not change the original array.

// map()

const numbers1 = [45, 4, 9, 16, 25];

const numbers2 = numbers1.map(myFunction);

console.log(numbers2) // [ 90, 8, 18, 32, 50 ]

function myFunction(value, index, array) {

  return value \* 2;

}

**filter()** method creates a new array with array elements that passes a test.

filtered\_number = numbers1.filter(filter\_function);

console.log(filtered\_number) //[ 45, 16, 25 ]

function filter\_function(value){

  return value > 10 // return only value which are greater than 18

}

**JavaScript Array reduce()**

The reduce() method runs a function on each array element to produce (reduce it to) a single value.

The reduce() method works from left-to-right in the array. The reduce() method does not reduce the original array.

Note that the function takes 4 arguments:

* The total (the initial value / previously returned value)
* The item value
* The item index
* The array itself

// reduce

sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {

  return total + value;

}

console.log(sum) // 99

sdafs