**JavaScript**

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

/////////////////////////////////

///////////////////////////////// /

**Animate**

Custom animation of a set of CSS properties

.animate( properties [, duration ] [, easing ] [, complete ] )

properties. An object of CSS properties and values that the animation will move towards

easing. which easing (aliviar) function to use for the transition

complete. A function to call once the animation is complete, called once per matched element

.animate( properties, options )

options. A map of additional options to pass to the method: duration, easing, queue, specialEasing, step, progress, complete, start, done, fail, always

( "#book" ).animate({

opacity: 0.25,

left: "+=50",

height: "toggle"

}, 5000, function() {

// Animation complete.

});

$( "#clickme" ).click(function() {

$( "#book" ).animate({

width: [ "toggle", "swing" ],

height: [ "toggle", "swing" ],

opacity: "toggle"

}, 5000, "linear", function() {

$( this ).after( "<div>Animation complete.</div>" );

});

});

Crece

<button id="**go**">&raquo; Run</button>

<div id="block">Hello!</div>

$( "#**go**" ).click(function() {

$( "#block" ).animate({

width: "70%",

opacity: 0.4,

marginLeft: "0.6in",

fontSize: "3em",

borderWidth: "10px"

}, 1500 );

Uno lo mueve a la izq, el otro, a la der

$( "#right" ).click(function() {

$( ".block" ).animate({ "left": "+=50px" }, "slow" );

});

$( "#left" ).click(function(){

$( ".block" ).animate({ "left": "-=50px" }, "slow" );

});

/////////////////////////////////

**After**

Insert content, specified by the parameter, after each element in the set of matched elements

<h2>Greetings</h2>

<div class="inner">Hello</div>

<div class="inner">Goodbye</div>

$( ".inner" ).after( "<p>Test</p>" );

Queda:

<h2>Greetings</h2>

<div class="inner">Hello</div>

<p>Test</p>

<div class="inner">Goodbye</div>

<p>Test</p>

<p>I would like to say: </p>

$( "p" ).after( document.createTextNode( "Hello" ) );

Queda:

I would like to say:

Hello

An element in the DOM can also be selected and inserted after another element:

$( ".container" ).after( $( "h2" ) );

/////////////////////////////////

**AddClass**

Does not replace a class

It simply adds the class

Is often used with .removeClass() to switch elements' classes from one to another:

$( "p" ).removeClass( "myClass noClass" ).addClass( "yourClass" );

$( "ul li" ).addClass(function( index ) {

return "item-" + index;

});

$( "p" ).last().addClass( "selected" );

Add the classes "selected" and "highlight":

$( "p" ).last().addClass( [ "selected", "highlight" ] );

if ( currentClass === "red" ) {

addedClass = "green";

}

//////////////////////////////////

$(selector).addClass(classname,function(index,currentclass))

/////////////////////////// ///

setInterval(function, milliseconds) repite la ejecución

<button onclick="setTimeout(**myFunction**, 3000)">Try it</button>

function **myFunction**() {

alert('Hello');

}

<button onclick="clearTimeout(myVar)">Stop it</button>

/////////////////////////// ///

navigator.platform

navigator.language

navigator.onLine

navigator.javaEnabled()

alert("Hello\nHow are you?");

if (confirm("Press a button!")) {

txt = "You pressed OK!";

} else {

txt = "You pressed Cancel!";

}

var person = prompt("Please enter your name", "Harry Potter");

/////////////////////////// ///

24 bits = 16,777,216 different "True Colors"

32 bits = 4,294,967,296 different "Deep Colors"

window.location.href (URL) of the current page

window.location.hostname domain name of the web host

window.location.pathname path and filename of the current page

window.location.protocol web protocol used (http: or https:)

window.location.assign() loads a new document

history.back() loads the previous URL in the history list

history.forward() loads the next URL in the history list

navigator.cookieEnabled

navigator.appName

navigator.appCodeName

navigator.product

navigator.appVersion

navigator.userAgent user-agent header sent by the browser to the server

The information from the navigator object can often be misleading, and should not be used to detect browser versions

/////////////////////////// ///

window.open() - open a new window

window.close() - close the current window

window.moveTo() - move the current window

window.resizeTo() - resize the current window

screen.width

screen.height

screen.availWidth

screen.availHeight

screen.colorDepth

screen.pixelDepth

/////////////////////////// ///

Browser Object Model (BOM)

The browser window (the browser viewport) is NOT including toolbars and scrollbars

Browser window (in pixels)

var w = window.innerWidth

|| document.documentElement.clientWidth

|| document.body.clientWidth;

var h = window.innerHeight

|| document.documentElement.clientHeight

|| document.body.clientHeight;

/////////////////////////// ///

Find the child you want to remove, and use its parentNode property to find the parent:

var **child** = document.getElementById("p1");

**child**.parentNode.removeChild(**child**);

var parent = document.getElementById("div1");

var **child** = document.getElementById("p1");

parent.replaceChild(para, **child**);

var paragraph = document.getElementsByTagName("p");

y = paragraph [1];

Change the background color of all <p> elements:

var **myCollection** = document.getElementsByTagName("p");

var i;

for (i = 0; i < **myCollection**.length; i++) {

**myCollection**[i].style.color = "red";

}

/////////////////////////// /

var **para** = document.createElement("p");

var node = document.createTextNode("This is new.");

**para**.appendChild(node);

var **element** = document.getElementById("div1");

var child = document.getElementById("p1");

**element**.insertBefore(**para**, child);

var elmnt = document.getElementById("p1");

elmnt.remove();

///////////////////////////

Use "click" instead of "onclick

A common error in DOM processing is to expect an element node to contain text

<title id="demo">DOM Tutorial</title>

var myTitle = document.getElementById("demo").firstChild.nodeValue;

or

var myTitle = document.getElementById("demo").childNodes[0].nodeValue;

document.body

document.documentElement - The full document

nodeName always contains the uppercase tag name of an HTML element

///////////////////////////

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

<h1 onclick="changeText(this)">Click on this text!</h1>

function changeText(id) {

id.innerHTML = "Ooops!";

}

document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {

document.getElementById("demo").innerHTML = Date();

}

/////////////////////////// /

document.getElementById("myImage").src = "landscape.jpg";

document.getElementById("p2").style.color = "blue";

<button type="button" onclick="document.getElementById('id1').style.color = 'red'">

**Cuadrado q se mueve en diagonal**

<div id ="container">

<div id ="**animate**"></div>

</div>

function myMove() {

var **elem** = document.getElementById("**animate**");

var pos = 0;

var **id** = setInterval(frame, 5);

function frame() {

if (pos == 350) {

clearInterval(**id**);

} else {

pos++;

**elem**.style.top = pos + "px";

**elem**.style.left = pos + "px";

}

}

}

/////////////////////////// /

var x = document.querySelectorAll("p.intro");

var x = document.forms["frm1"];

var text = "";

var i;

for (i = 0; i < x.length; i++) {

text += x.elements[i].value + "<br>";

}

document.getElementById("demo").innerHTML = text;

Never use document.write() after the document is loaded. It will overwrite the document

/////////////////////////// /

document.anchors

.applets

.baseURI

.body

.cookie

.doctype

.documentElement

.documentMode

.documentURI

.domain

.domConfig

.embeds

.forms

.head

.images

.implementation

.inputEncoding

.lastModified

.links

.readyState

.referrer

.scripts

.strickErrorChecking

.title

.URL

/////////////////////////// /

element.innerHTML =

element.attribute =

element.style.property =

element.setAttribute(attribute, value)

document.createElement(element)

.removeChild

.appendChild

.replaceChild

.write

document.getElementById(id).onclick = function(){code}

/////////////////////////// /

The W3C DOM standard is separated into 3 different parts:

Core DOM - standard model for all document types

XML DOM - standard model for XML documents

HTML DOM - standard model for HTML documents

The innerHTML property can be used to get or change any HTML element, including <html> and <body>

var x = document.getElementsByTagName("p");

var x = document.getElementsByClassName("intro");

var x = document.querySelectorAll("p.intro");

var x = document.forms["frm1"];

var text = "";

var i;

for (i = 0; i < x.length; i++) {

text += x.elements[i].value + "<br>";

}

document.getElementById("demo").innerHTML = text;

///////////////////////////

async makes a function return a Promise

await makes a function wait for a Promise

async function myFunction() {

return "Hello";

}

myFunction().then(

function(value) {myDisplayer(value);},

function(error) {myDisplayer(error);}

);

///////////////////////////

function myDisplayer(some) {

document.getElementById("demo").innerHTML = some;

}

function **getFile**(**myCallback**) {

let req = new XMLHttpRequest();

req.open('GET', "mycar.html");

req.onload = function() {

if (req.status == 200) {

**myCallback**(this.responseText);

} else {

**myCallback**("Error: " + req.status);

}

}

req.send();

}

**getFile**(myDisplayer);

///////////////////////////

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

When you pass a function as an argument, remember not to use parenthesis

You can pass a whole function instead:

setTimeout(function() { aFunction("I love You !!!"); }, 3000);

setInterval(myFunction, 1000);

function myFunction() {

let **d** = new Date();

document.getElementById("demo").innerHTML=

**d**.getHours() + ":" +

**d**.getMinutes() + ":" +

**d**.getSeconds();

}

///////////////////////////

Browsers should not download more than two components in parallel

An alternative is to use defer="true" in the script tag. The defer attribute specifies that the script should be executed after the page has finished parsing, but it only works for external scripts

Si el js va en la misma página que el html, usar:

window.onload = function () {

};

Avoid using the with keyword. It has a negative effect on speed. It also clutters up JavaScript scopes.

The with keyword is not allowed in strict mode

///////////////////////////

Statements or assignments that can be placed outside the loop will make the loop run faster

var i;

var l = arr.length;

for (i = 0; i < l; i++) {

}

If you expect to access a DOM element several times, access it once, and use it as a local variable

Don't create new variables if you don't plan to save values

Replace:

var fullName = firstName + " " + lastName;

document.getElementById("demo").innerHTML = fullName;

With:

document.getElementById("demo").innerHTML = firstName + " " + lastName;

///////////////////////////

If you use a named index, when accessing an array, JavaScript will redefine the array to a standard object.

After the automatic redefinition, array methods and properties will produce undefined or incorrect results

JavaScript objects, variables, properties, and methods can be undefined.

In addition, empty JavaScript objects can have the value null.

This can make it a little bit difficult to test if an object is empty.

You must test for not undefined before you can test for not null:

if (typeof myObj !== "undefined" && myObj !== null)

///////////////////////////

You must use a "backslash" if you must break a statement in a string:

var x = "Hello \

World!";

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does not support arrays with named indexes.

In JavaScript, arrays use numbered indexes

///////////////////////////

switch statements use strict comparison

All numbers in JavaScript are stored as 64-bits Floating point numbers (Floats).

All programming languages, including JavaScript, have difficulties with precise floating point values:

var x = 0.1;

var y = 0.2;

var z = x + y // the result in z will not be 0.3

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

var z = (x \* 10 + y \* 10) / 10; // z will be 0.3

///////////////////////////

It is a good habit to assign default values to arguments.

function myFunction(x, y) {

if (y === undefined) {

y = 0;

}

}

End Your Switches with Defaults

default:

day = "Unknown";

///////////////////////////

Crear una clase

let myCar1 = new Car("Ford", 2014);

Evite variables globales, new, ==, y eval()

Initialize variables when you declare them

Use [] instead of new Array()

Use /()/ instead of new RegExp()

Use function (){} instead of new Function()

Beware of Automatic Type Conversions, numbers can accidentally be converted to strings or NaN (Not a Number)

When doing mathematical operations, JavaScript can convert numbers to strings

///////////////////////////

With arrow functions there are no binding of this

With arrow functions the this keyword always represents the object that defined the arrow function

///////////////////////////

If you have parameters, you pass them inside the parentheses:

hello = (val) => "Hello " + val;

if you have only one parameter, you can skip the parentheses as well

hello = val => "Hello " + val;

///////////////////////////

hello = function() {

return "Hello World!";

}

hello = () => {

return "Hello World!";

}

hello = () => "Hello World!";

///////////////////////////

Not Real Constants

The keyword const is a little misleading.

It does NOT define a constant value. It defines a constant reference to a value.

Because of this, we cannot change constant primitive values, but we can change the properties of constant objects.

///////////////////////////

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

arName = "Volvo";

alert(carName);

var carName;

Variables defined with let are hoisted to the top of the block, but not initialized.

Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared.

///////////////////////////

Global variables defined with the var keyword belong to the window object:

var carName = "Volvo";

// code here can use window.carName

let carName = "Volvo";

// code here cannot use window.carName

///////////////////////////

both have Global Scope:

var x = 2; // Global scope

let x = 2; // Global scope

///////////////////////////

both have Function Scope:

function myFunction() {

var carName = "Volvo"; // Function Scope

}

function myFunction() {

let carName = "Volvo"; // Function Scope

}

///////////////////////////

var i = 5;

for (var i = 0; i < 10; i++) {

// some statements

}

// Here i is 10

let i = 5;

for (let i = 0; i < 10; i++) {

// some statements

}

// Here i is 5

///////////////////////////

Variables declared with the var keyword cannot have Block Scope.

Variables declared inside a block {} can be accessed from outside the block.

{

var x = 2;

}

// x CAN be used here

Variables declared with the let keyword can have Block Scope.

Variables declared inside a block {} cannot be accessed from outside the block:

{

let x = 2;

}

// x can NOT be used here

///////////////////////////

You can use strict mode in all your programs. It helps you to write cleaner code, like preventing you from using undeclared variables

"use strict";

In strict mode, the this keyword refers to the object that called the function

If the object is not specified, functions in strict mode will return undefined and functions in normal mode will return the global object (window)

///////////////////////////

Hoisting is JavaScript's default behavior of moving declarations to the top.

Variables defined with let and const are hoisted to the top of the block, but not initialized.

Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared

JavaScript only hoists declarations, not initializations.

To avoid bugs, always declare all variables at the beginning of every scope

///////////////////////////

If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.

carName = "Volvo";

In "Strict Mode", undeclared variables are not automatically global

With JavaScript, the global scope is the complete JavaScript environment

In HTML, the global scope is the window object

Do NOT create global variables unless you intend to.

Your global variables (or functions) can overwrite window variables (or functions)

///////////////////////////

try {

adddlert("Welcome guest!");

}

catch(err) {

document.getElementById("demo").innerHTML = err.message;

}

///////////////////////////

**exec**() searches a string for a specified pattern, and returns the found text as an object.

If no match is found, it returns an empty (null) object

/**e**/.**exec**("The best things in life are free!");

S: true

///////////////////////////

**test**() searches a string for a pattern, and returns true or false

var patt = /**e**/;

patt.**test**("The best things in life are free!");

S: true

///////////////////////////

[abc] Find any of the characters

[0-9] Find any of the digits

(x|y) Find any of the alternatives

\d Find a digit

\s Find a whitespace character

\b Find a match at the beginning of a word

\uxxxx Find the Unicode character specified by the hexadecimal number

n+ Matches any string that contains at least one n

n\* Matches any string that contains zero or more occurrences of n

n? Matches any string that contains zero or one occurrences of n

///////////////////////////

var str = "\nIs th\n**is** it?";

var patt1 = /^is/**m**;

S: is

m Perform multiline matching

///////////////////////////

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced

var str = "Visit W3Schools!";

var n = str.search("W3Schools");

S: 6

case-insensitive

var n = str.search(/w3schools/i);

g Perform a global match (find all matches rather than stopping after the first match)

var str = "**Is** this all there **is**?";

var patt1 = /is/**g**;

var result = str.match(patt1);

S: is,is

//////////////////

When a mathematical function or operation in JavaScript cannot return a specific

number, it returns the value NaN instead

//////////////////////////////

isNaN checks if the value is a number, if not tries to convert it (\*), and then checks if the resulting value is NaN.

isNaN("45.3"); // false

isNaN(NaN); // true

isNaN(1); // false: 1 is a number

isNaN(""); // false: converted to 0, which is a number

isNaN(" "); // false: converted to 0, which is a number

isNaN("45.3"); // false: string representing a number, converted to 45.3

isNaN("1.2e3"); // false: string representing a number, converted to 1.2e3

isNaN("hello"); // true : conversion fails, no digits at all

isNaN(undefined); // true : converted to NaN

isNaN(); // true : converted to NaN (implicitly undefined)

isNaN(function(){}); // true : conversion fails

isNaN({}); // true : conversion fails

isNaN([1, 2]); // true : converted to "1, 2", which can't be converted to a number

/////////////////////////////

To properly check if a value is null, compare it with the strict equality operator

var a = null;

a === null; // true

//////////////////////////

var myInteger = 12; // 32-bit number (from -2,147,483,648 to 2,147,483,647)

var myLong = 9310141419482; // 64-bit number (from -9,223,372,036,854,775,808 to

9,223,372,036,854,775,807)

var myFloat = 5.5; // 32-bit floating-point number (decimal)

var myDouble = 9310141419482.22; // 64-bit floating-point number

//////////////////////////////////////////////////////

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

Scripts can be placed in the <body>, or in the <head>, or in both

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

<button type="button" onclick="document.write(5 + 6)">Try it</button>

Ventana emergente

window.alert(5 + 6);

console.log(5 + 6);

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

/////////////////////////////////////////////////////

JavaScript ignores multiple spaces

var x;

comentarios: // /\* \*/

It is case sensitive

var \_x = 2;

var $myMoney = 5;

\*\* exponenciación

!=

&&

||

!

//////////////////////////////

var x = 5;

var y = 2;

var z = x % y;

S: 1

querySelector lets you find elements with rules that can't be expressed with getElementById and getElementsByClassName

**Tipos de dato**

var length = 16; // Number

var lastName = "Johnson"; // String

var x = {firstName:"John", lastName:"Doe"}; // Object

**Vector:**

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

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

In JavaScript, the data type of null is an object

typeof undefined // undefined

typeof null // object

null === undefined // false

null == undefined // true

typeof {name:'John', age:34} // Returns "object"

typeof [1,2,3,4] // Returns "object" not "array"

typeof function myFunc(){} // Returns "function"

/////////////////////////////////////////////////////////// /

function myFunction(a, b) {

return a \* b;

}

var x = myFunction(4, 3);

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

It is executed when:

When an event occurs (when a user clicks a button)

When it is invoked (called) from JavaScript code

Automatically (self invoked)

Accessing a function without () will return the function object instead of the function result.

This will return the function definition

function toCelsius(f) { return (5/9) \* (f-32); }

instead of the function result:

document.getElementById("demo").innerHTML = toCelsius;

////////////////////////////////////////////////////////////////////

let te permite declarar variables limitando su alcance (scope) al bloque, declaración, o expresión donde se está usando

var define una variable global o local en una función sin importar el ámbito del bloque

Objects can contain many values.

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

Acceder a un O:

person.lastName;

person["lastName"];

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

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.

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

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

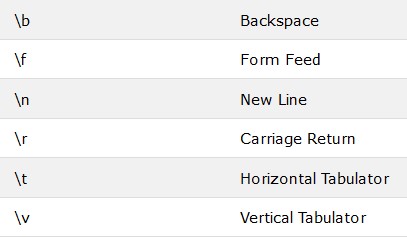
var z = new Boolean(); // Declares z as a Boolean object

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Las cadenas pueden ser con comillas dobles o simples

Cadena.length;



var firstName = "John";

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

var firstName = new String("John");

The new keyword complicates the code. This can produce some unexpected results

var x = new String("John");

var y = new String("John");

// (x == y) is false because x and y are different objects

/////////////////////////////////////////////////////////////////

JavaScript treats primitive values as objects when executing methods and properties.

Returns the index of (the position of) the first occurrence of a specified text in a string:

indexOf()

JavaScript counts positions from zero.

Cannot take powerful search values (regular expressions)

indexOf()

Empieza desde el final

lastIndexOf()

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

Buscar el texto locate empezando desde el caracter 15:

str.indexOf("locate", 15);

Searches a string for a specified value and returns the position of the match.

Cannot take a second start position argument:

search()

slice(start, end) el final es descontando 1.

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.

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

substring(start, end) cannot accept negative indexes.

The second parameter specifies the length of the extracted part

**////////////////////////////////////////////////////**

**str.replace("Microsoft", "W3Schools");**

**Is case sensitive**

**Case insensitive:**

**str.replace(/MICROSOFT/i, "W3Schools");**

**To replace all matches:**

**str.replace(/Microsoft/g, "W3Schools");**

**text1.toUpperCase();**

**text1.concat(" ", text2);**

**All string methods return a new string.**

**They don't modify the original string.**

**The trim() method is not supported in Internet Explorer 8 or lower.**

**If you need to support IE 8, you can use replace() with a regular expression instead:**

**var str = " Hello World! ";**

**alert(str.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, ''));**

**o**

**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());**

**///////////////////////////////////////////**

**charAt(position) returns the character at a specified index**

**charCodeAt(position) returns the unicode of the character**

**Property access [ ] tiene muchas complicaciones, mejor no usarlo:**

**var str = "HELLO WORLD";**

**str[0];**

**S: H**

**Convertir a vector:**

**var txt = "a,b,c,d,e"; // String**

**txt.split(","); // Split on commas**

**txt.split(" "); // Split on spaces**

**txt.split("|"); // Split on pipe**

**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**

**/////////////////////////////////////////////////////////////////////////////////**

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

**can be written with scientific (exponent) notation:**

**var x = 123e5; // 12300000**

**var y = 123e-5; // 0.00123**

**JavaScript Numbers are Always 64-bit Floating Point**

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

**var x = 999999999999999; // x will be 999999999999999**

**var y = 9999999999999999; // y will be 10000000000000000**

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

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

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

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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

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

**var x = 10;**

**var y = 20;**

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

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

**var x = "100";**

**var y = "10";**

**var z = x / y; // z will be 10**

**////////////////////////////////////////**

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

x will be NaN (Not a Number):

var x = 100 / "Apple";

isNaN(x);

returns true because x is Not a Number:

typeof NaN;

returns "number" :

typeof Infinity;

returns "number":

var x = new Number(500);

var y = new Number(500);

// (x == y) is false because objects cannot be compared

//////////////////////////////////////////////////////////

JavaScript treats primitive values as objects when executing methods and properties

returns a string 9.66e+0:

x.toExponential(2);

Cantidad de decimales

returns 9.66:

var x = 9.656;

x.toFixed(2);

returns 9.7:

var x = 9.656;

x.toPrecision(2);

returns a number 123:

var x = 123;

x.valueOf();

///////////////////////////////////////////////////////////////

Number()

Number(new Date("2017-09-30"));

S:

1506729600000:

parseFloat()

parseInt()

parseInt("10");

S:

10

parseInt("10.33");

S:

10

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

var x = Number.MIN\_VALUE;

////////////////////////////////////////

var cars = [

"Saab",

"Volvo",

"BMW"

];

var cars = new Array("Saab", "Volvo", "BMW");

document.getElementById("demo").innerHTML = cars[0];

El vector completo:

document.getElementById("demo").innerHTML = cars;

Array Elements Can Be Objects:

myArray[0] = Date.now;

myArray[1] = myFunction;

myArray[2] = myCars;

var x = cars.length;

var y = cars.sort();

Accessing the Last Array Element:

var last = fruits[fruits.length - 1];

text = "<ul>";

fruits.forEach(myFunction);

text += "</ul>";

function myFunction(value) {

text += "<li>" + value + "</li>";

}

**////////////////////////////////////////////////////////////////////**

**fruits.push("Lemon");**

**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**

**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**

**In JavaScript, objects use named indexes (índices con nombre)**

**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**

**var points = new Array(); // Bad**

**var points = []; // Good**

**The new keyword only complicates the code. It can also produce some unexpected results**

**////////////////////////////////////////////////////////////////////////////**

**How to Recognize an Array, JavaScript operator typeof returns "object":**

**Array.isArray(fruits); // returns true, is not supported in older browsers**

**You can create your own isArray() function:**

**function isArray(x) {**

**return x.constructor.toString().indexOf("Array") > -1;**

**}**

**it returns true if the object prototype contains the word "Array".**

**fruits.join(" \* ");**

**pop() method removes the last element, returns the value that was "popped out"**

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

**fruits.push("Kiwi");**

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

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

**Since JavaScript arrays are objects, elements can be deleted by using:**

**delete fruits[0];**

**Using delete may leave undefined holes in the array. Use pop() or shift() instead.**

**splice() add new items to an array: posición, cuantos serán removidos, elementos especificos a insertar. Returns an array with the deleted items**

**concat() method creates a new array by merging (concatenating) existing arrays, does not change the existing arrays. It always returns a new array**

**var myChildren = arr1.concat("Peter");**

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

**var citrus = fruits.slice(3);**

**var citrus = fruits.slice(1, 3);**

**/////////////////////////////////////////**

myi = parseInt(readLine());

myd = parseFloat(readLine());

mys = readLine();

fruits.sort(); No funciona con números

fruits.reverse();

Organizar números:

points.sort(function(a, b){return a - b});

Descendentemente:

points.sort(function(a, b){return b - a});

Un método más exacto:

for (i = points.length -1; i > 0; i--) {

j = Math.floor(Math.random() \* i)

k = points[i]

points[i] = points[j]

points[j] = k

}

the highest number in an array:

Math.max.apply(null, arr);

Más rápido:

function myArrayMax(arr) {

var len = arr.length;

var max = -Infinity;

while (len--) {

if (arr[len] > max) {

max = arr[len];

}

}

return max;

}

El mínimo:

function myArrayMin(arr) {

var len = arr.length;

var min = Infinity;

while (len--) {

if (arr[len] < min) {

min = arr[len];

}

}

return min;

}

Sorting Object Arrays

var cars = [

{type:"Volvo", year:2016},

{type:"Saab", year:2001},

{type:"BMW", year:2010}

];

A compare function to compare the property values:

cars.sort(function(a, b){return a.year - b.year});

Comparing string properties:

cars.sort(function(a, b){

var x = a.type.toLowerCase();

var y = b.type.toLowerCase();

if (x < y) {return -1;}

if (x > y) {return 1;}

return 0;

});

////////////////////////////

function Person(initialAge) {

// Add some more code to run some checks on initialAge

if (initialAge < 0) {

console.log("Age is not valid, setting age to 0.");

this.age = 0;

} else {

this.age = initialAge;

}

this.amIOld = function() {

// Do some computations in here and print out the correct statement to the console

if (this.age < 13) {

console.log("You are young.");

} else if (this.age >= 13 && this.age < 18) {

console.log("You are a teenager.");

} else {

console.log("You are old.");

}

};

this.yearPasses = function(){

// Increment the age of the person in here

this.age++;

};

}

function main() {

var T=parseInt(readLine());

for(i=0;i<T;i++){

var **age**=parseInt(readLine());

var p=new Person(**age**);

p.amIOld();

for(j=0;j<3;j++){

p.yearPasses();

}

p.amIOld();

console.log("");

}

}

///////////////////////

Var. By default a declared yet not initialized variable has undefined value

If the variable is accessed before declaration, it evaluates to undefined

Hoisting: Izar

Any function given to the setTimeout function will be executed asynchronously, when the main thread is not busy anymore

Never trust the delay given to the setTimeout function. Your code could be executed much later

Pitfall: Escollo

In JavaScript null is also considered an object

Al revisar los tipos de dato, revise primero si la variable en null

Si la variable no tiene var, es global

In situations where you want to also return true for object or functions:

console.log((bar !== null) && ((typeof bar === "object") || (typeof bar === "function")));

Is bar an object:

console.log((bar !== null) && (typeof bar === "object"));

////////////////////////////////////////// /

var numbers = [45, 4, 9, 16, 25];

numbers.**forEach**(myFunction);

function myFunction(value, index, array) { }

**The map()**

Creates a new array by performing a function on each array element.

Does not execute the function for array elements without values.

Does not change the original array.

var numbers2 = numbers1.**map**(myFunction);

function myFunction(value, index, array) { }

**filter()**

Creates a new array with array elements that passes a test

var numbers = [45, 4, 9, 16, 25];

var over18 = numbers.**filter**(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

//////////////////////////////////////////

**The reduce()**

Runs a function on each array element to produce (reduce it to) a single value.

Works from left-to-right in the array.

Does not reduce the original array.

The sum of all numbers in an array:

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

var sum = numbers1.**reduce**(myFunction);

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

return total + value;

}

**every()**

Check if all array values pass a test.

Check if all array values are larger than 18:

var numbers = [45, 4, 9, 16, 25];

var allOver18 = numbers.**every**(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

**some()**

Check if some array values pass a test.

This example check if some array values are larger than 18:

var numbers = [45, 4, 9, 16, 25];

var someOver18 = numbers.**some**(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

**indexOf()**

Searches an array for an element value and returns its position.

The first item has position 0, the second item has position 1, and so on.

Search an array for the item "Apple":

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

var a = fruits.**indexOf**("Apple");

array.indexOf(item, start)

item Required. The item to search for.

start Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

Returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

**lastIndexOf()**

But returns the position of the last occurrence of the specified element

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

var a = fruits.**lastIndexOf**("Apple");

**find()**

Returns the value of the first array element that passes a test function

**findIndex()**

Method returns the index of the first array element that passes a test function

/////////////////////////////////////////

The Boolean value of:

0 (zero) is false

"" (empty string) is false

undefined is false

null is false

NaN is false

false is false

var x = new Boolean(false);

var y = new Boolean(false);

(x == y)

Is false because objects cannot be compared

///////////////////////////

!== not equal value or not equal type

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

switch (new Date().getDay()) {

case 6:

text = "Today is Saturday";

break;

case 0:

text = "Today is Sunday";

break;

default:

text = "Looking forward to the Weekend";

}

case 4:

case 5:

text = "Soon it is Weekend";

break;

If multiple cases matches a case value, the first case is selected.

Switch cases use strict comparison (===)

The values must be of the same type to match

//////////////////////////

for (i = 0, len = cars.length, text = ""; i < len; i++) { }

for (; i < len; i++) { }

for (; i < len; ) { }

var person = {fname:"John", lname:"Doe", age:25};

var text = "";

var x;

for (x in person) {

text += person[x];

}

Loops through the properties of an object

var cars = ["BMW", "Volvo", "Mini"];

var x;

for (x of cars) {

document.write(x + "<br >");

}

Loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists…

///////////////////////////////

while (i < 10) {

text += "The number is " + i;

i++;

}

do {

text += "The number is " + i;

i++;

}

while (i < 10);

/////////////////////////////

String(123)

S: returns a string from a number literal 123

Number("3.14") // returns 3.14

Number(" ") // returns 0

Number("") // returns 0

Number("99 88") // returns NaN

parseFloat() Parses a string and returns a floating point number

parseInt() Parses a string and returns an integer

Number(false) // returns 0

Number(true) // returns 1

/////////////////////////////





/////////////////////////////

Para debuguear poner debugger; en el navegador, en Sources, el programa se parará ahí

Abajo en Watch > + > escribir algo q se desea buscar

Se pueden poner breakpoints

Si se pone un break point dentro de una función, abajo en Call Stack se pude ver q funciones se ha llamado hasta ahí

Para que los cambios en el navegador queden en VS Code: Sources > FileSystem > + > escoger el archivo > guardar

Debuguear en VS Code:

En VS Code instalar la extensión Debugger for Chrome > a la derecha, abrir el ícono del bicho

Arriba en el triángulo verde > Add Configuration > Chrome > guardar

Se abre launch.json > cambiar para q quede: “webRoot”: “${workspaceFolder}/public”

A la derecha, escoger el ícono del bicho > arriba, darle al triángulo verde

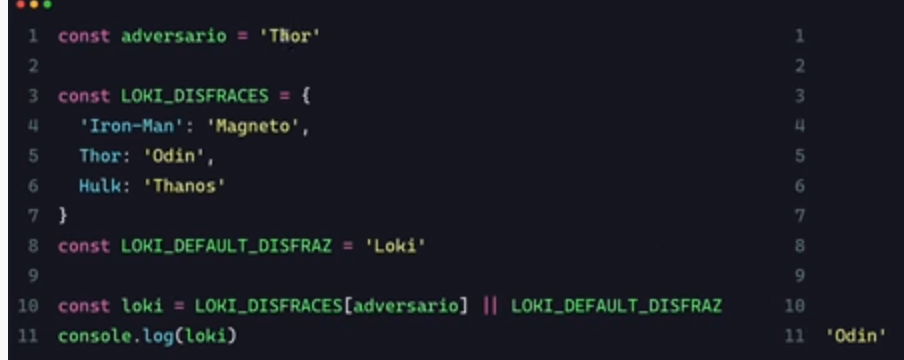
Ahora puede poner break points en VS Code

Breakpint condicional:

En VS Code > click en un breakpoint > Edit breakpoint > poner una condición, por ej: usernameInput.value === ‘’ así parará si el textbox está vacío

<https://www.youtube.com/watch?v=AX7uybwukkk>

/////////////////////////////



/////////////////////////////



/////////////////////////////

/////////////////////////////

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

hackerrank voy en el

5

<https://www.youtube.com/watch?v=5XyzLfPBpZs>

<https://www.codingame.com/playgrounds/347/javascript-promises-mastering-the-asynchronous/quick-quiz>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/RegExp>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions>

<https://www.w3schools.com/jsref/jsref_obj_regexp.asp>

<https://stackoverflow.com/questions/48955444/reshape-string-inserting-n-at-every-n-characters>

<https://www.google.com/search?client=firefox-b-d&q=javascript+tricky+questions>

<https://www.toptal.com/javascript/interview-questions>

<https://dmitripavlutin.com/javascript-hoisting-in-details/>

<https://github.com/getify/You-Dont-Know-JS/issues/767#issuecomment-227946671>

<https://dmitripavlutin.com/simple-explanation-of-javascript-closures/>

Ya

Javascript querySelector CDATA in xml file

Javascript querySelector vs. getElementById [closed]

<https://stackoverflow.com/search?q=querySelector+javascript>