**Introducción**

Entonces, ¿quieres aprender a programar en JavaScript? ¡Genial!

Este curso será un viaje bastante largo, pero si llega al final, podrá leer, comprender y, por supuesto, escribir aplicaciones y programas JavaScript. Estas nuevas habilidades pueden ayudarlo en su trabajo actual o permitirle alcanzar nuevas oportunidades profesionales en un mercado de TI en constante crecimiento. Comencemos esta aventura sin más preámbulos y descubramos qué es JavaScript.

**Cómo comunicarse con la computadora**

Comencemos con algo obvio: las computadoras están en todas partes. Es casi seguro que esté leyendo este curso en una computadora; tu teléfono es una computadora; encontrarás computadoras en televisores y otros electrodomésticos, como tu aspiradora automática, que utiliza su computadora y numerosos sensores en su interminable misión de erradicar el polvo de tu vida. Estamos rodeados de computadoras. Las computadoras se utilizan en ciencia, medicina, banca y negocios. Sospechamos que hoy en día sería difícil encontrar algún ámbito de la vida en el que los ordenadores no participen activamente.

Usamos computadoras principalmente porque son capaces de realizar ciertas tareas incomparablemente más rápido y con mayor precisión que las personas. Sin embargo, las computadoras no pueden adivinar qué tipo de tareas tenemos en mente o cómo ayudarnos a realizarlas. Tenemos que decirles eso. ¿Cómo? Es mejor hacerlo de manera similar a cuando se pasa información a otras personas, en otras palabras, utilizando un lenguaje que sea comprensible para ambas partes. Utilizando ese lenguaje, escribimos un programa, una solución formalizada a nuestro problema, que puede ser ejecutada por la computadora.

Desafortunadamente, un lenguaje que sea directamente comprensible por una computadora será absolutamente ilegible para un ser humano normal. Sería una secuencia de instrucciones extrañas, escritas en forma numérica, que se refieren a componentes de computadora que ni siquiera sabíamos que existían (y, francamente, no necesariamente tenemos que saberlo). Esta forma de comunicación, que se remonta a los inicios de la informática, se utiliza hoy en día muy raramente y solo en situaciones muy concretas.

Entonces, para ayudar en la comunicación con la computadora, se inventaron lenguajes de programación que son algo similares a los lenguajes naturales (es decir, los que se usan para la comunicación entre personas). Hay miles de lenguajes de programación y difieren en su propósito (además de los lenguajes de propósito general, hay muchos especializados, por ejemplo, lenguajes diseñados solo para cálculos estadísticos), nivel de abstracción (en términos simples: cuanto mayor es el nivel del lenguaje, menos necesitamos saber sobre el hardware en el que se ejecuta el programa), facilidad de uso, efectividad de los programas escritos en ellos, etc.

Independientemente del idioma que elija, algunas cosas permanecen iguales mientras escribe el programa. En primer lugar, una etapa importante, y probablemente la más difícil, de este proceso es definir correctamente el problema que queremos resolver. Solo entonces intentamos encontrar la solución óptima, que finalmente presentaremos en forma de programa escrito en el idioma elegido.

Entonces, antes de comenzar a explicarle algo a la computadora, en otras palabras, escribir un programa, debe comprender exactamente lo que desea lograr y cómo desea lograrlo. En segundo lugar, la solución que proponemos y escribimos en forma de programa debe ser 100% inequívoca: la computadora no puede adivinar nada.

Un ejemplo simple de un campo ligeramente diferente: en algún momento de su vida, probablemente compró un mueble que requirió ensamblaje. Montarlo es un problema con el que usted, el comprador, se ha visto agobiado. Para que pueda hacer frente a esta tarea, recibirá un conjunto de instrucciones que lo guiarán a lo largo de todo el proceso. En este momento, está actuando como intérprete, utilizando un programa que le permitirá completar la tarea. El éxito de su misión depende de la calidad de estas instrucciones, si son precisas e inequívocas y no proceden de otro mueble. Al final, puede resultar que no hayas construido los muebles de tus sueños, sino una construcción surrealista de otra dimensión.



https://www.youtube.com/watch?v=6X6HqL67YmU

Para que las instrucciones sean buenas, quien las desarrolle debe saber exactamente qué deben ilustrar, en qué orden deben realizarse ciertas acciones, en qué etapas es más fácil confundir algo, etc. Y, por supuesto, deben saber qué efecto se va a lograr al final.

**Constantes**

El propósito principal de una constante es erradicar la posibilidad de cambiar accidentalmente un valor almacenado en ella. Esto es importante cuando tenemos algunos valores que realmente nunca deberían cambiar. Los ejemplos típicos de constantes son rutas a recursos, tokens y otros datos que nunca cambian durante la vida útil del script.

Pero las constantes también se pueden usar como subresultados en cálculos o en otros lugares donde la información que se recopiló o calculó no cambiará más. El uso de una constante, además de evitar que un valor se cambie por error, permite que el motor de JavaScript optimice el código, lo que puede afectar su rendimiento.

2.1.1.14 Variabes - the keyword var cd.

pay attention: prestar atención

In most cases: en la mayoría de los casos

Let’s make: hagamos

And most of all: y sobre todo

We can perform: podemos realizar

goes beyond the scope: va más allá del alcance

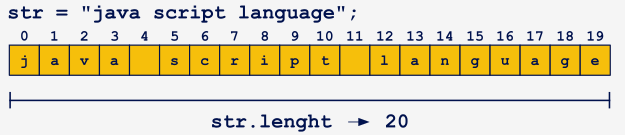
Let's take a look at a simple example: echemos un vistazo a un ejemplo simple

Metodo

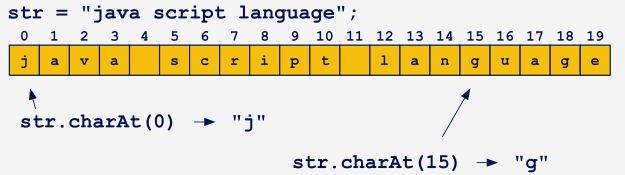
Un método es un tipo especial de función que pertenece a un objeto. Los objetos son tipos de datos complejos, que pueden constar de muchos valores (almacenados en propiedades) y métodos. Si desea llamar al método de un objeto, escriba el nombre del método después de un punto. ¿Esto te recuerda algo? Esta es exactamente la notación que usa cuando llama a console.log. El objeto de la consola tiene muchos otros métodos además del método de registro, como time y timeEnd (que se pueden usar para medir el tiempo).

Los métodos y propiedades de cadena de uso común (es decir, valores con nombre relacionados con el objeto) son:

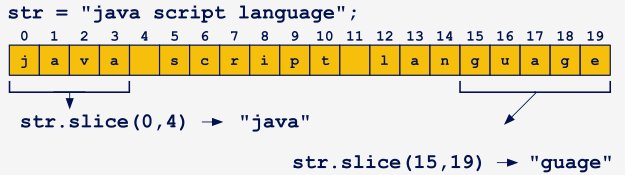
lenght: propiedad, devuelve el número de caracteres en una cadena;



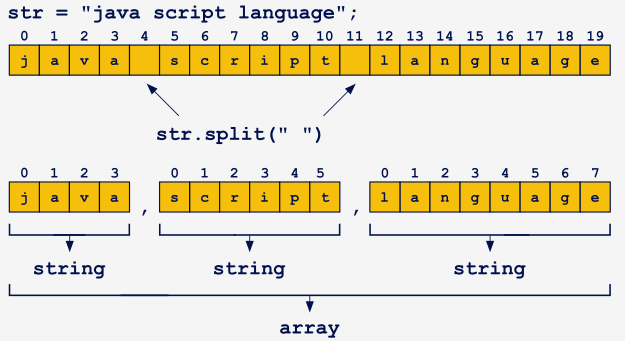
charAt (índice): método, devuelve el carácter en la posición "índice" en la cadena (los índices comienzan desde 0);



slice (beginIndex, [opcional] endIndex): método, devuelve una nueva cadena que se crea a partir de los caracteres entre beginIndex (incluido) y endIndex (excluido); si se omite endIndex, entonces la nueva cadena es desde beginIndex hasta el final de la cadena;



split (separator, [opcional] limit): método, divide la cadena en subcadenas cada vez que se encuentra un separador en esa cadena, y devuelve una matriz de esas subcadenas (diremos algunas palabras sobre las matrices en un momento), mientras que una opción limit limita el número de subcadenas agregadas a la lista.



Carry out: realizar

So let's improve: así que mejoremos

If you can add new fields to an existing object, can you also delete them? Of course you can: the “delete” operator is used for this.

Shifted: desplazado

ternario

El último de los operadores discutidos es bastante inusual, porque es el único operador que usa tres operandos. Es un operador condicional. Según el valor del primer operando (verdadero o falso), se devuelve el valor del segundo o tercer operando, respectivamente. Este operador se usa con mayor frecuencia para colocar uno de los dos valores en la variable dependiendo de una determinada condición. Volveremos al operador cuando analicemos el condicional si, pero aquí proporcionaremos solo un ejemplo simple de su uso. Los tres operandos están separados entre sí por? (el primero del segundo) y: (el segundo del tercero).

console.log(true ? "Alice" : "Bob"); // -> Alice

console.log(false ? "Alice" : "Bob"); // -> Bob

Cada uno de estos operandos puede ser una expresión que debe calcularse. En el siguiente ejemplo, el primer operando es una comparación de dos números usando un operador de comparación. El resultado de la comparación será falso, que será utilizado por el operador condicional (ternario). Aquí llegamos a un problema importante sobre la precedencia de los operadores y el orden de ejecución. En un momento, diremos algunas palabras más al respecto.

let name = 1 > 2 ? "Alice" : "Bob";

console.log(name); // -> Bob

and so on: etc

ventana emergente o modales o cuadro de dialogo

// Costo del seguro

const INSURANCE\_COST = 2.99;

// Costo de envío

let shippingCost = 9.99;

// ¿Es válida la orden?

let isOrderValid = true;

// Edad del usuario

let userAge = 22;

// Puntos

let points = 400;

// Valor del carrito

let cartValue = 199;

// Tiene un código de promoción

let hasPromoCode = false;

// Tiene la aprobación de los padres

let hasParentsApproval = false;

// Agregar seguro

let addInsurance = true;

/\*\* calculate shipping cost\*/

if ((userAge > 65) || (userAge >= 21 && (hasPromoCode || cartValue > 300 || points > 500))) {

shippingCost = 0;

} else if (userAge < 21 && hasParentsApproval) {

shippingCost = shippingCost - 5;

} else if (userAge < 21) {

isOrderValid = false;

}

/\*\* take account of insurance \*/

if (isOrderValid && addInsurance && !hasPromoCode) {

shippingCost += INSURANCE\_COST;

}

/\*\* show message \*/

if (isOrderValid) {

console.log("Costo de envío: " + shippingCost);

} else {

console.log("Cannot order if under 21");

}

La declaración if ... else ... if - continúa

En el siguiente ejemplo más extenso, podemos ver el uso de ifs en cascada con elses, y también condiciones lógicas complejas. Siéntase libre de jugar con los valores asignados a las variables para ver cómo cambian los resultados.

Para resumir lo que está sucediendo, podemos analizar cada caso por separado:

* Si userAge es menor de 21 y hasParentsApproval es falso, el pedido no es válido;
* Si userAge es menor que 21 pero hasParentsApproval se establece en true, el costo de envío se reducirá en 5;
* Si la edad del usuario es de 65 años o más, el costo de envío se reduce a cero;
* Si userAge es menor de 65, pero mayor o igual a 21 Y uno de los siguientes:

- hasParentsApproval es igual a verdadero;

- cartValue es mayor que 300;

- points es superior a 500;

el costo de envío (shippingCost) se reduce a cero.

En cualquier otro caso, el costo permanece en el valor predeterminado.

Después de todo esto, hacemos otra verificación:

* Si addInsurance se establece en verdadero;
* Y además orderIsValid;
* Y hasPromoCode no es verdadero, entonces agregamos INSURANCE\_COST al shippingValue.

Al final, mostramos el costo de envío si el pedido es válido y el mensaje si no lo es.

Tómese su tiempo con este ejemplo y juegue con los valores para comprender qué está sucediendo y cómo.

All in all: considerándolo todo

for (initialization; condition; increment) {

block of code

}

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

console.log(i);

}

Notación científica

33e3 => 33 \* (10 ^ 3) => 33000

18.65e6 => 18.65 \* (10 ^ 6) => 18650000

3.3e1 => 3.3 \* (10 ^ 1) => 33

4.5808747e7 => 4.5808747 \* (10 ^ 7) => 45808747

E6 => 10 \* 10 \*10 \*10 \*10 \*10 => 1 000 000

10 ^ -3 = 10

For of (para array)

for (variable of array) {

block of code

}

For in (para objetos)

for (let key in user) {

console.log('${key} -> ${user[key]}');

}

If(a == 0) ⬄ ¡a

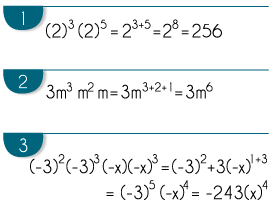
**Funciones: variables locales**

Intentemos hacer un pequeño cambio en nuestro programa calculando la temperatura media. ¿Recuerdas qué son las variables locales? Así es como llamamos a las variables que se declaran y usan en un alcance limitado y no son visibles en todo el programa, lo que significa que solo podemos usarlas dentro de ese alcance particular. Las variables declaradas con la palabra clave let son locales dentro del bloque de código (es decir, dentro del rango limitado por corchetes), mientras que las variables declaradas con la palabra clave var son locales dentro del bloque de funciones. Entonces, si declara una variable dentro de un bloque de funciones, ya sea usando let o var, solo será visible (y utilizable) dentro del bloque de funciones. Esto es muy útil, porque normalmente las variables que usa dentro de una función no son necesarias fuera de ella.

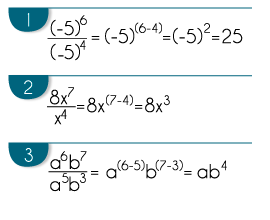
En nuestro código, un ejemplo de dicha variable es suma. Aunque la hemos declarado fuera de la función getMeanTemp (es una variable global), solo nos referimos a ella dentro de la función. Por tanto, una declaración global es innecesaria. Pongámoslo en orden, declarando suma localmente.

El comportamiento del programa es el mismo, pero el código ha ganado algo de claridad. La variable de suma ahora es local, solo se puede usar dentro de la función getMeanTemp (que es suficiente para nosotros, porque no era necesaria para nada fuera de la función). En general, debemos esforzarnos por mantener el código de la función lo más separado posible del contexto circundante, entre otras cosas, evitando el uso de variables globales en su interior. En nuestro ejemplo, hay dos variables más de este tipo: temperaturas y meanTemp. Este último, meanTemp, se usa dentro de la función para almacenar y devolver el resultado calculado. Tratemos con eso.

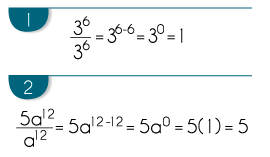
**Producto de potencias de la misma base**



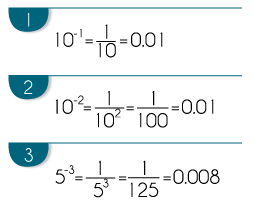
**División de potencias de la misma base**



**Potencia con exponente cero**



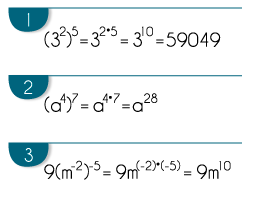
**Potencia con exponente negativo**



x-m = 1 / xm

2-2 = 1 / 22 = .25

**Potencia de potencias**



a-1 = 1/a1

a0 = 1

and frankly speaking: y francamente hablando

but there are many functions: pero hay muchas funciones

**Lenguajes naturales y errores de comunicación**

Imagina que estás organizando una fiesta junto al lago para tus amigos. Como organizador, explicas a todos cómo llegar, pero como siempre, en tales situaciones, alguien se pierde. Ya casi están allí, pero luego envían un mensaje de texto pidiendo consejos adicionales. Envías una respuesta, en la que les dices que giren a la derecha en el primer camino después de dejar el bosque, y luego conduzcan otros 500 metros. Como todavía está en camino, pídales que esperen en el lugar. El mensaje correcto podría verse así: After leaving the forest, turn right into the first path, and drive 500m. Wait on the spot. (después de dejar el bosque, gire a la derecha en el primer camino y conduzca 500 m. Espere en el lugar).

√ (x) = x(½)

Confianza limitada

Los programas no se ejecutan en el vacío. Por lo general, durante su ejecución, hay interacciones con los usuarios (por ejemplo, ingresando los datos necesarios para calcular ciertos valores) u otros programas o sistemas (por ejemplo, descargando datos del servidor). El comportamiento tanto de los usuarios como de otros sistemas debe tratarse con precaución, y no podemos asumir que el usuario proporcionará datos en el formato que requerimos, o que el servidor de datos siempre funcionará. Estas situaciones inesperadas también serán fuentes de errores en nuestro programa. Y aunque no dependen directamente de nosotros, es nuestra responsabilidad anticiparnos a situaciones potencialmente peligrosas. Si, por ejemplo, escribimos una calculadora en la que el usuario ingresa sus valores, entonces probablemente deberíamos verificar si el divisor no es cero antes de hacer la división. Teóricamente, el usuario debe saber que no dividimos entre cero, sino que somos responsables de la estabilidad del programa. No crea al usuario ni a otros sistemas. Predecir qué puede salir mal y verificar los datos recibidos antes de usarlos en su programa.

Escribamos un fragmento de código que le pedirá que ingrese dos números. Luego queremos mostrar el resultado de dividir el primer número por el segundo:

let sX = prompt("Enter the first number");

let sY = prompt("Enter the second number");

let x = Number(sX);

let y = Number(sY);

if (Number.isFinite(x) && Number.isFinite(y) && y !== 0) {

console.log(x / y);

} else {

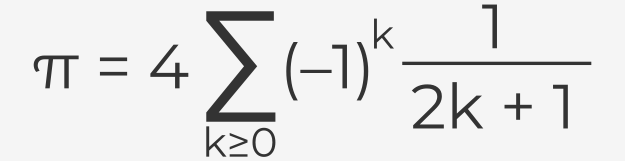
console.log("incorrect arguments");

}

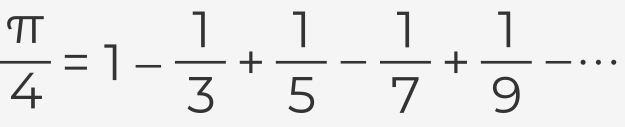
<https://edube.org/learn/jse1-1-0/debugging>

Measuring code execution time - continued

Suppose we want to calculate the approximate value of the number π. There are many methods that allow this, one of which is to use the Leibniz formula:



which can be expanded into the series:



The value of π calculated in this way is approximate, but it is more accurate the longer the series is (i.e. the larger the value of k we use). Higher accuracy will of course involve a larger number of operations to be performed and therefore will affect the time of program execution. Let’s check what an example program written in JavaScript might look like, which would allow us to perform such calculations:

let part = 0;

for (let k = 0; k < 10000000; k++) {

part = part + ((-1) \*\* k) / (2 \* k + 1);

}

let pi = part \* 4;

console.log(pi); // -> 3.1415925535897915

The variable part will contain a partial result, which will be modified in each iteration of the for loop. The loop will be executed ten million times (i.e. k will take values from 0 to 999999). The most time-consuming part will be the execution of the for loop, because in each iteration, operations such as summing, multiplying, dividing, and exponentiation are performed.

Let's check how much time it takes for the program to execute this piece of code. For this purpose, we will use the methods console.time and console.timeEnd.

let part = 0;

console.time('Leibniz');

for (let k = 0; k < 10000000; k++) {

part = part + ((-1) \*\* k) / (2 \* k + 1);

}

console.timeEnd('Leibniz'); // -> Leibniz: 456.60498046875 ms

let pi = part \* 4;

console.log(pi); // -> 3.1415925535897915

With console.time, we indicate where to start the time measurement, while with console.timeEnd we end the measurement, and the result is displayed on the console at this point (the result you get will obviously differ from the one in the example). The time is given in milliseconds. In the calls of the console.time and console.timeEnd methods, we can specify a string (in the example it is 'Leibnitz') that will identify our stopwatch in case we use many of them in our program.

Let's look inside the for loop. In each iteration, the number -1 is raised to the power of k. Exponentiation is quite a time-consuming operation, so we can suspect that it strongly affects the speed of our program (especially as we do it ten million times). If the base of the exponentiation is the number -1, we will always get -1 or 1 as a result, depending on whether the exponent is odd or even. In this case, we can replace exponentiation with a conditional instruction that checks if k is even (divisible by 2) or odd and returns 1 or -1 respectively.

let part = 0;

console.time('Leibniz');

for(let k = 0; k < 10000000; k++) {

part = part + (k % 2 ? -1 : 1) / (2 \* k + 1);

}

console.timeEnd('Leibniz'); // -> Leibniz: 175.5458984375 ms

let pi = part \* 4;

console.log(pi);

As you can see, even such a small change allows us to more than double the speed of the program! Using the console.time and console.timeEnd methods allows us to analyze our code for performance. If we have the impression that something works too slowly, but we do not know which piece of code is responsible for this, we can perform measurements, locate the problem and, optionally, try to optimize the code. As we said before, there are many tools that also help us in this. Some of them are built into the Developer Tools integrated with the browser, but often the methods shown are enough to perform basic tests.

Try to test both the solutions in your local environment, and see what differences in times you get.