JavaScript Fundamentals

Arturo Aguado Arnold

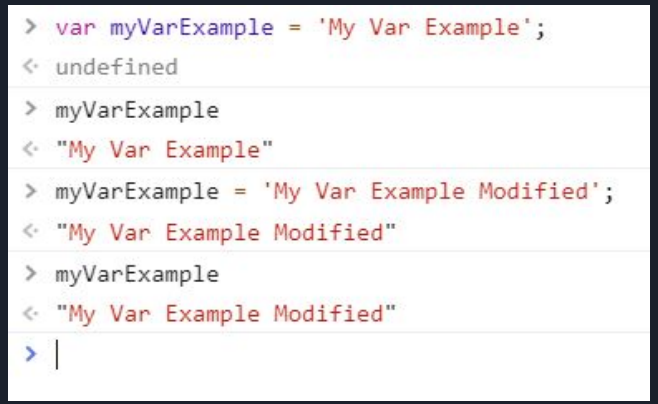
Un poquito de historia...

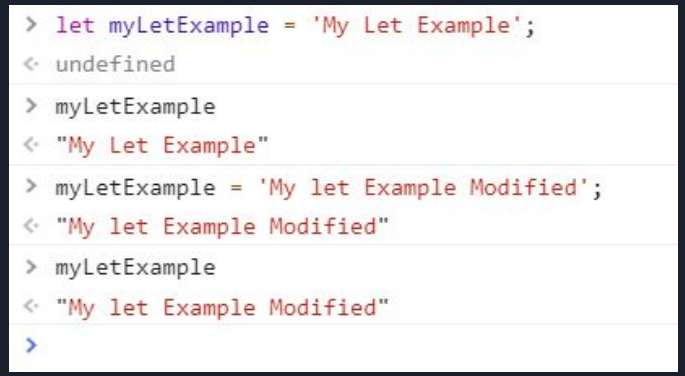
- Creado en 1995 por *Brendan Eich* inicialmente con el nombre *Mocha*, en *Netscape*.
- En 1997 los autores propusieron llamarlo JavaScript para que fuera adoptado como estándar por la *European Computer Manufacturers Association* (ECMA).
- ECMAScript (1996): Especificación de lenguaje de programación. Basado en JavaScript.
- Junio 2015 - ECMAScript 6 - ECMAScript 2015
- Junio 2016 - ECMAScript 7 - ECMAScript 2016
    - Módulos para organización de código, clases, arrow functions, promesas...

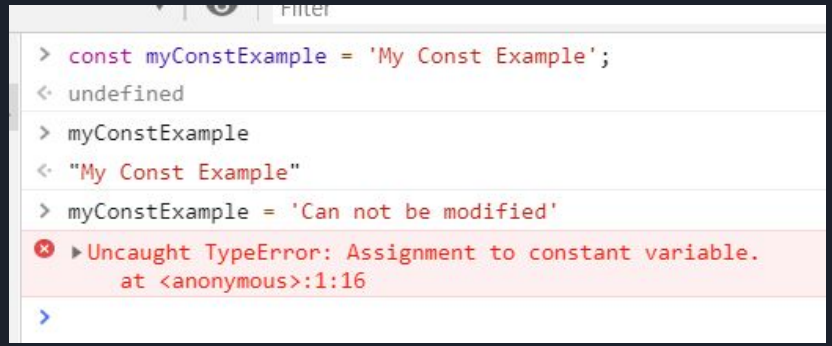
Para los curiosos…
https://es.wikipedia.org/wiki/JavaScript
https://es.wikipedia.org/wiki/ECMAScript

Variables

- Var (viejo javascript)
- Let (nuevo javascript)
- Const (nuevo javascript)







Variables

- El nombre puede contener letras, números o los símbolos $ ó _ .
- No puede comenzar por números.

```
let userName;
let test123;
```

```
let $ = 1; // declared a variable with the name "$"
let _ = 2; // and now a variable with the name "_"
```

Tampoco se pueden poner como nombre algunas de estas palabras reservadas, pero no os preocupéis que se aprenden con el tiempo.

Variables

```
1  let message;
```

Reserva una posición de memoria y la denomina “message” para poder acceder a ella más fácilmente.

```
3  message = 'Hello'; // store the string
```

Asigna un valor a la posición de memoria apuntada por la variable “message”.

```
1  let user = 'John', age = 25, message = 'Hello';
```

También se pueden crear múltiples variables en una sola línea y asignarles un valor en su creación.

Variables

Se le puede asignar un nuevo valor en cualquier momento.

```
let message;

message = 'Hello!';

message = 'World!'; // value changed
```

Se le puede asignar el valor de una variable a otra variable.

```
let hello = 'Hello world!';

let message;

// copy 'Hello world' from hello into message
message = hello;

// now two variables hold the same data
```

DATA-TYPES

```
> let name = 'Pepe', age = 30, married = true, hasPets;
< undefined
> let person = {
      name: name,
      age: age,
      married: married,
      hasPets: hasPets
  }
< undefined
> person
< ▼{name: "Pepe", age: 30, married: true, hasPets: undefined}
    age: 30
    hasPets: undefined
    married: true
    name: "Pepe"
  ▶ __proto__: Object
>
```

Tipos primitivos:

- Number.
- String.
- Boolean
- Null
- Undefined
- Symbol: Para crear identificadores únicos para objetos.

No primitivos:

- Object: A diferencia de los demás tipos, el tipo objeto es algo especial. Con él podemos crear colecciones complejas.

DATA-TYPES

¿Y si queremos saber el tipo de una variable en tiempo de ejecución?

Tenemos el operador typeof que nos devuelve una string con el tipo de la variable.

typeof variable ó type(variable)

```
> typeof name
"string"
> typeof(age)
"number"
> typeof person;
"object"
```

```
> typeof married;
"boolean"
> typeof hasPets
"undefined"
>
```

DATA-TYPES

En Javascript, las variables pueden tener cualquier tipo de valor y cambiar en tiempo de ejecución:

```
1  // no error
2  let message = "hello";
3  message = 123456;
```

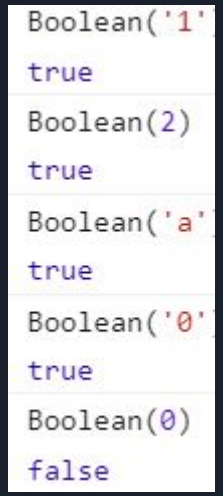
```
1  let n = 123;
2  n = 12.345;
```

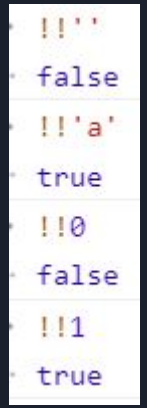
Las operaciones matemáticas son seguras, para ello tenemos los tipos *Inifinity* y *NaN*, por lo que nunca se parará la ejecución del proceso debido a un error de cálculo.

```
> 1 / 0
< Infinity
```

```
> "This is not a number" / 2
< NaN
```

Conversión de tipos





Operadores

Términos:

- Operando: Elemento al que se le aplica el operador.
- Unario: Se llama el operador si aplica sólo a un operando.
- Binario: Se llama el operador si aplica a más de un operando.

```
> // Unario
  let x = 5;
  x = -x;
  console.log('x tiene el valor', x);

  x tiene el valor -5
```

```
> // Binario
  let y = 5, z = 10;
  console.log(y + z);

  15
```

Operadores

- Asignación: =
- Suma / Resta: + / -
- Multiplicación: *
- División: /
- Exponente: **
- Módulo: %
- Incremento / Disminución: ++ / --

```
> let counter = 0;
  console.log(++counter);
  console.log(counter--);
  console.log(counter);
  1
  1
  0
```

Operadores

Algunas características especiales de los operadores de JavaScript...

Concatenación de strings:

```
> // String Concatenation.
  let literal = "my" + "String";
  let literal2 = "1" + 2 + 3;
  let literal3 = 1 + 2 + '3';

  console.log(literal);
  console.log(literal2);
myString
123
```

```
> console.log(literal3);
33
```

Operadores

Algunas características especiales de los operadores de JavaScript...

Conversión numérica:

```
> // Sin efecto en números.
  let number1 = 1
  let number2 = -2
  console.log(+number1);
  console.log(+number2);
  1
  -2
```

```
> console.log(+true);
  console.log(+false);
  console.log(+"1");
  console.log(+"0");
  console.log(+"");
  console.log(+"a");
  1
  0
  1
  0
  0
  NaN
```

```
> let apples = "2";
  let oranges = "3";

  console.log(apples + oranges);
```

```
> console.log(+apples + +oranges);
  5
```

Comparadores

Tienen como resultado un tipo boolean, true o false:

- Mayor / menor que: a > b / a < b
- Mayor / menor o igual que: a > b / a < b
- Igual que (sin tipo): “1” == 1
- Igual que (con tipo) “1” === 1
- Distinto de (sin tipo) “1” != 1
- Distinto de (con tipo) “1” !== 1

Cuidado con los tipos!!!

```
> let a = 0;
  console.log( Boolean(a) ); // false

  let b = "0";
  console.log( Boolean(b) ); // true

  console.log(a == b); // true!

  console.log(a === b) // false!
```

Operadores condicionales

Evalúan una condición que retorna un valor de tipo boolean, true / false. Ejecutará las condiciones en orden hasta que se cumpla una. Cuando una condición se cumple entra en su respectivo bloque, ejecuta sus instrucciones y no pasará por las siguientes condiciones.

```
> if (condicion) {

  // ...

  } else if (condicion2) {

  // ...

  } else {

  // ...

  }
```

SWITCH

No es un operador condicional pero puede sustituir al operador “IF” en muchas situaciones.

```
let myValue = 2 + 2;

switch(myValue) {
    case 3:
        console.log('Te has quedado corto');
        break;
    case 4:
        console.log('Exactamente!');
        break;
    case 5:
        console.log('Te has pasado!');
        break;
    default:
        console.log('No tengo respuesta para el valor introducido');
}
```

Operadores lógicos

OR (||)

```
let var1 = null;
let var2 = 'Pepe';
let var3 = 'Sin nombre';

let var4 = var1 || var2 || var3;
```

AND (&&)

```
let v1 = true;
let v2 = 'nombre';
let v3 = false;
let v4 = v1 && v2 && v3;
let v5 = v1 && v2
```

NOT (!)

```
> let ex1 = true;
  let ex2 = !ex1;
```

Bucles

Los bucles ejecutan las instrucciones englobadas entre las llaves mientras la condición devuelva un valor booleano true.

```
while(condicion)
```

```
> let i = 0;

  while(i < 4) {
      console.log(i);
      i++;
  }
```

```
> let z = 0;
  do {
  console.log(z);
  z++;
  } while (z < 3);
```

Bucles

- FOR es el bucle más usado de todos.
- La declaración tiene tres partes:
  - Inicio: Declaraciones de variables usadas por el bucle.
  - Condición: Mientras la condición retorne un valor boolean true seguirá iterando.
  - Paso: Después de ejecutar la última instrucción del bucle ejecutará estas instrucciones.

```
for(inicio ; condicion ; paso)
```

```
for (let i=0 ; i < 5 ; i++) {
    console.log(i);
}
```

Bucles

¿Y si quisiéramos interrumpir una determinada iteración o saltar a la siguiente?

```
> let i = 0;
  while (i < 5) {
      i++;
      break;
  }
```

```
> let i = 0;
  while (i < 5) {
      i++;
      if (i === 3) {
          continue;
      }
      console.log(i);
  }
```

Funciones

- Nos permiten agrupar funcionalidad.
- Evitan la duplicidad de código.
- Son los principales bloques de un programa.
- Las funciones permiten definir un comportamiento para usarlo más de una vez.

```
function showMessage() {
    console.log('This function is awesome!');
}

showMessage();
showMessage();
```

```
This function is awesome!
This function is awesome!
```

Funciones. Variables.

Las variables que se declaran dentro de un bloque, son sólo accesibles dentro de ese bloque. Podemos distinguir entonces entre variables *locales* y *externas*.

```
function showMessage() {
    let message = 'This is taking shape...'
    console.log(message);
}

showMessage();
console.log(message);
```

```
This is taking shape...
Uncaught ReferenceError: message is not defined
    at <anonymous>:10:13
>
```

```
let message = 'This is taking shape...';
function showMessage() {
    console.log(message);
}

showMessage();
console.log(message);
```

```
This is taking shape...
This is taking shape...
```

Funciones. Parámetros o argumentos.

- Mediante los parámetros o argumentos se puede pasar información a una función.
- Los argumentos pueden tener un valor por defecto.

```
> function showMessage(from, text) {
    console.log(from + ': ' + text);
}

showMessage('Arturo', '¡Hola!');
showMessage('Arturo', '¿Qué tal?');
```

```
Arturo: ¡Hola!
Arturo: ¿Qué tal?
```

```
> function showMessage(from, text = 'No se ha recibido texto') {
    console.log(from + ': ' + text);
}

showMessage('Arturo');
```

```
Arturo: No se ha recibido texto
```

Arrays

- Referencia.
- Colecciones de cualquier tipo: number, string, boolean,arrays, objects, functions…

¿Cómo se declaran?

```
let myArray1 = new Array();
let myArray2 = [];
let myArray3 = [1, 2, 3, 4, 5]
```

¿Y si quiero cambiar un valor?

```
myArray3[3] = 10;
```

¿Cómo obtener un determinado elemento?

```
myArray3[3];
```

¿Cómo se obtiene el número total de elementos?

```
myArray3.length
```

Arrays

Métodos:

- length: Devuelve el número total de elementos del array.
- push: Añade un elemento al final del array.
- pop: Elimina el ultimo elemento del array.
- shift: Elimina el primer elemento del array.
- unshift: Añade un elemento al principio del array.

Arrays

Hemos visto cómo obtener y modificar un elemento del array pero… ¿Y si queremos leer o modificar cada uno de los elementos del array?

```
let fruitsList = ["Apple", "Orange", "Pear"];

for (let i = 0; i < fruitsList.length; i++) {
  console.log( fruitsList[i] );
}
```

Los arrays también se pueden iterar de la siguiente manera:

```
for (let fruit of fruitsList ) {
  console.log( fruit );
}
```