

JavaScript

Índice

Índice

Variables

Declaración de variables

var

let

const

Tipos

Tipo de datos primitivos

Números

Cadenas de texto (strings)

Booleanos

Tipos de datos compuestos

Arrays

Objects

Funciones

Declaración de funciones

Keyword Function

Arrow Functions

Función de una sola línea

Operadores

Operadores aritméticos

Operadores de asignación

Operadores lógicos

Operadores de comparación

Operadores de concatenación

Operadores de acceso

Acceso a elementos de un objeto

Acceso a elementos de un array

Estructuras de control

If

If else

If else if

Switch

Estructuras de repetición

While

Do while

For

Estructuras de control de flujo

Break

Continue

Callbacks

Funciones sobre arrays

Función push

Función pop

Función slice

Función forEach

Función map

Función filter

Operador spread

Variables

Declaración de variables

En JavaScript, existen 3 formas de declarar variables:

var

```
var variable = valor;
```

Esta forma de declarar las variables hace que éstas sean globales. Es decir, la variable que declaremos así, podrá ser accedida desde cualquier parte del código. Por ejemplo:

```
for (var i = 0; i < 10; i++) {  
  console.log(i);  
}  
console.log("El valor de i es: " + i); // Output: El valor de i es: 10
```

Por esta razón, no es recomendable usar la forma de declaración de variables con var, y en general utilizaremos la forma siguiente.

let

```
let variable = valor;
```

Esta forma de declarar las variables, se asegura de que sean locales, y tengan un "scope" limitado a un bloque de código. Por ejemplo:

```
for (let i = 0; i < 10; i++) {  
  console.log(i);  
}  
console.log("El valor de i es: " + i); // Output: El valor de i es: undefined
```

Utilizar las variables declaradas con let nos permite evitar errores. Por ejemplo, si declaramos una variable con el mismo nombre dentro de una función, y luego dentro de otra, podremos acceder a ella sin problemas.

const

```
const variable = valor;
```

Esta es una forma particular de declarar variables, que prohíbe que estas sean modificadas. Por ejemplo:

```
const PI = 3.1416;  
PI = 3.14; // Error
```

const es útil a la hora de crear variables que sabemos que no deben ser modificadas nunca y evitar problemas a posteriori.

Tipos

JavaScript es un lenguaje de tipado dinámico. Por ello, no existe un tipo de dato para las variables explícito, sino que los tipos son inferidos automáticamente.

Tipo de datos primitivos

Números

```
let numero = 1; // Enteros  
let numero2 = 1.5; // Floats  
let numero3 = 1.5e2; // Notación científica: 1.5 x 10^2  
let numero4 = 1.5e-2; // Notación científica: 1.5 x 10^-2  
let numero5 = 0xff; // Hexa: 255  
let numero6 = 0b1010; // Binario: 10  
let numero7 = 0o777; // Octal: 503
```

Cadenas de texto (strings)

```
let cadena = "Hola"; // Comillas dobles
let cadena2 = 'Hola'; // Comillas simples
let cadena3 = `${cadena}
Mundo`; // Comillas que aceptan multilinea y variables
```

Booleanos

```
let booleano = true;
let booleano2 = false;
```

Tipos de datos compuestos

Arrays

```
let array = [1, 2, 3];
let array2 = [1, 2, 3, "Hola", true]; // Pueden tener distintos datos
let array3 = [1, 2, 3, "Hola", true, [1, 2, 3]]; // Pueden tener arrays dentro de arrays
```

Objects

Es importante destacar que los objetos de JavaScript no son objetos como los que se conocen en otros lenguajes. En JavaScript, los objetos son una colección de pares clave-valor, como por ejemplo un diccionario de Python, un array asociativo de PHP o un mapping en Solidity. No necesitan tener que instanciarse desde una clase, ya que javascript usa otro paradigma de objetos llamado prototipado.

```
let objeto = {
  nombre: "Juan",
  apellido: "Perez",
  edad: 30,
  // Los objetos pueden tener objetos dentro
  direccion: {
    calle: "Calle falsa",
    numero: 123
  }
  // y también funciones
  saludar: function() {
    console.log("Hola");
  }
}
```

Funciones

Las funciones en JavaScript, como en cualquier otro lenguaje, son procedimientos o bloques de código que se ejecutan cuando son invocados.

Declaración de funciones

Se pueden declarar funciones de dos formas. A efectos cotidianos, son equivalentes, pero hay una sutil diferencia: al usar el keyword `function` se puede hacer uso del keyword `this` que es usado en prototipado.

Keyword Function

Se pueden declarar funciones mediante el uso de la palabra reservada del sistema, `function`.

```
function nombreDeLaFuncion(parametros) {
  // Código de la función
}
```

De forma equivalente, se puede hacer también

```
const nombreDeLaFuncion = function (parametros) {
  // Código de la función
}
```

Arrow Functions

En JavaScript, a partir de ES6 se pueden declarar funciones de la siguiente forma.

```
const nombreDeLaFuncion = (parametros) => {  
  // Código de la función  
};
```

Como mencionamos antes, este tipo de declaración es equivalente salvo por el uso del `this`

Función de una sola línea

En algunos casos, es más fácil declarar una función con una sola línea de código. Esto es posible únicamente si la función tiene una única línea de código. Además, podemos omitir el `return` de la función, ya que JavaScript asume que la función devuelve el resultado de esa línea.

```
const multiplicarPor2 = (x) => x * 2;
```

Operadores

Los operadores son un conjunto de funciones que nos permiten realizar operaciones matemáticas, comparaciones, procesos sobre strings, etc.

Operadores aritméticos

Son los operadores que nos permiten realizar operaciones aritméticas, es decir, operaciones sobre números.

```
// Suma  
let resultado = 3 + 2; // 5  
// Resta  
let resultado = 3 - 2; // 1  
// Multiplicación  
let resultado = 3 * 2; // 6  
// División  
let resultado = 3 / 2; // 1.5  
// Exponente  
let resultado = 3 ** 2; // 9  
// Modulo (resto de la división)  
let resultado = 3 % 2; // 1
```

Operadores de asignación

Son los operadores que nos permiten asignar valores y cambiar el contenido de las variables.

```
let x = 1;  
x += 2; // x = x + 2  
x -= 2; // x = x - 2  
x *= 2; // x = x * 2  
x /= 2; // x = x / 2  
x %= 2; // x = x % 2  
x++; // x = x + 1  
x--; // x = x - 1
```

Operadores lógicos

Son los operadores que nos permiten realizar operaciones lógicas, es decir, operaciones sobre booleanos.

```
let x = true;  
let y = false;  
// AND (&&)  
let z = x && y; // false  
// OR (||)  
let w = x || y; // true
```

```
// NOT (!)
let v = !x; // false
```

Operadores de comparación

Son los operadores que nos permiten realizar comparaciones entre valores.

```
let x = 1;
let y = 2;
// Menor que (<)
let z = x < y; // true
// Mayor que (>)
let w = x > y; // false
// Menor o igual que (≤)
let v = x ≤ y; // true
// Mayor o igual que (≥)
let u = x ≥ y; // false
// Igualdad (==)
let t = x == y; // false
// Desigualdad (≠)
let s = x ≠ y; // true
```

Operadores de concatenación

Son los operadores que nos permiten concatenar strings, es decir, pegar uno o más strings para formar uno más largo con el contenido de todos.

```
let x = "Hola";
let y = " mundo";
// Concatenación (+)
let z = x + y; // "Hola mundo"
// Concatenación en la asignación (+=)
let x += y; // x = x + y
// Cadenas multilinea (`)
let w = `${x} ${y}`; // "Hola mundo"
let v = `Hola mundo.
    Esto es una cadena multilinea.`; // "Hola mundo.\nEsto es una cadena multilinea."
```

Operadores de acceso

Son los operadores que nos permiten acceder a los elementos de un objeto o un array.

Acceso a elementos de un objeto

```
let x = {
  nombre: "Juan",
  apellido: "Perez",
  edad: 30,
};

// Acceso a una propiedad (.)
let y = x.nombre; // "Juan"

// Acceso a una propiedad (["nombre"])
let z = x["nombre"]; // "Juan"
```

Acceso a elementos de un array

```
let x = [1, 2, 3, 4, 5];
x[0]; // 1
x[1]; // 2
x[2] = 10; // x = [1, 2, 10, 4, 5]
x[3]; // 4
```

Estructuras de control

If

```
if (condición) {  
    // Código a ejecutar si la condición es verdadera  
}
```

If else

```
if (condición) {  
    // Código a ejecutar si la condición es verdadera  
} else {  
    // Código a ejecutar si la condición es falsa  
}
```

If else if

```
if (condición1) {  
    // Código a ejecutar si la condición1 es verdadera  
} else if (condición2) {  
    // Código a ejecutar si la condición2 es falsa  
} else {  
    // Código a ejecutar si ninguna de las condiciones es verdadera  
}
```

Switch

```
switch (variable) {  
    case valor1:  
        // Código a ejecutar si la variable es igual al valor1  
        break;  
    case valor2:  
        // Código a ejecutar si la variable es igual al valor2  
        break;  
    default:  
        // Código a ejecutar si ninguna de las condiciones es verdadera  
        break;  
}
```

Estructuras de repetición

While

```
while (condición) {  
    // Código a ejecutar mientras la condición es verdadera  
}
```

Do while

```
do {  
    // Código a ejecutar mientras la condición es verdadera  
} while (condición);
```

For

```
for (let i = 0; i < 10; i++) {  
    // Código a ejecutar 10 veces  
}
```

Estructuras de control de flujo

Break

```
while (true) {
  // Código a ejecutar
  if (condición) {
    // Código a ejecutar si la condición es verdadera
    break; // El código se detiene acá
  }
}
```

Continue

```
while (true) {
  // Código a ejecutar
  if (condición) {
    // Código a ejecutar si la condición es verdadera
    continue; // El código continúa con la siguiente vuelta
  }
}
```

Callbacks

Los callbacks son funciones que se pasan como parámetros a otras funciones.

```
const saludoInformal = (nombre) => {
  console.log(`Hola ${nombre}`);
};

const saludoFormal = (nombre) => {
  console.log(`Bienvenido ${nombre}`);
};

const saludo = (nombre, callback) => {
  callback(nombre);
};

saludo("Juan", saludoInformal);

saludo("Roberto", saludoFormal);
```

Funciones sobre arrays

Los arrays tienen una serie de métodos que nos permiten realizar operaciones sobre ellos, sin tener que iterarlos con algún bucle, o facilitan alguna tarea. La mayoría de estas utilizan callbacks.

Función push

Esta función agrega un elemento al final del array.

```
let x = [1, 2, 3, 4, 5];
x.push(6); // x = [1, 2, 3, 4, 5, 6]
```

Función pop

Esta función elimina el último elemento del array.

```
let x = [1, 2, 3, 4, 5];
x.pop(); // x = [1, 2, 3, 4]
```

Función slice

Esta función devuelve una copia del array en un rango especificado por parámetros.

```
let x = [1, 2, 3, 4, 5];
// El primer parámetro indica el índice del primer elemento a devolver.
// El segundo parámetro indica el índice del primer elemento que no se incluirá.
x.slice(2, 4); // [3, 4, 5]
x.slice(0, 1); // [1]
x.slice(0, -2); // [1, 2, 3]

// Podemos omitir el segundo parámetro para indicar que se devuelvan todos los elementos a partir del primer parámetro.
x.slice(2); // [3, 4, 5]
x.slice(0); // [1, 2, 3, 4, 5]
```

Función forEach

Esta función nos permite aplicar una función (callback) a cada elemento del array y modificarlo.

```
let x = [1, 2, 3, 4, 5];
x.forEach((elemento, indice) => {
  // Código a ejecutar para cada elemento del array
});
```

Podemos recibir únicamente el elemento del array.

```
let x = [1, 2, 3, 4, 5];
x.forEach((elemento) => elemento * 2); // x = [2, 4, 6, 8, 10]
```

Función map

Esta función nos permite aplicar una función (callback) a cada elemento del array y devuelve un nuevo array con los resultados de la aplicación de la función a cada elemento.

```
let x = [1, 2, 3, 4, 5];
x.map((elemento, indice) => {
  // Código a ejecutar para cada elemento del array
});
```

Podemos recibir únicamente el elemento del array. Por ejemplo:

```
let x = [1, 2, 3, 4, 5];
let y = x.map((elemento) => elemento + 2); // y = [3, 4, 5, 6, 7]
```

Función filter

Esta función nos permite aplicar una función booleana (callback) a cada elemento del array y devuelve un nuevo array con los elementos que cumplen la condición de la función.

```
let x = [1, 2, 3, 4, 5];
x.filter((elemento, indice) => {
  // Código a ejecutar para cada elemento del array
});
```

Podemos recibir únicamente el elemento del array. Por ejemplo:

```
let x = [1, 2, 3, 4, 5];
let y = x.filter((elemento) => elemento % 2 === 0); // y = [2, 4]
```

Operador spread

Este operador (...) permite "desarmar" un array en una serie de elementos, de forma que podemos, por ejemplo, pasarlo como parámetro a una función.

```
let x = [1, 2, 3, 4, 5];
let y = [7, 8, ... x, 10, 12, 16]; // y = [7, 8, 1, 2, 3, 4, 5, 10, 12, 16]
```


Ejemplo de uso:

```
const sumar = (a, b, c) => {  
  return a + b + c;  
}  
  
let x = [1, 2, 3];  
let z = sumar(...x); // z = 6
```