



Somos un **ecosistema** de desarrolladores de software

Introducción a Java Script



```
<!-- _____ BEGIN NAVIGATION  
h">
```

```
">Home</a></li>  
.html">Home Events</a></li>  
enu.html">Multiple Column Men  
<a href="#" class="current"
```

```
utton-header.html">Tall But  
logo.html">Image Logo</a></  
href="tall-logo.html">Ta
```

```
f="#">Carousels</a>
```

```
th-slider.html">Variab  
lider.html">Testimoni
```

Tipos de datos

Básicos

Tipo de dato	Descripción	Ejemplo básico
Number	Valor numérico (enteros, decimales, etc...)	42
BigInt	Valor numérico grande	1234567890123456789n
String	Valor de texto (cadenas de texto, caracteres, etc...)	'MZ'
Boolean	Valor booleano (valores verdadero o falso)	true
undefined	Valor sin definir (variable sin inicializar)	undefined
Function	Función (función guardada en una variable)	function() {}
Symbol	Símbolo (valor único)	Symbol(1)
Object	Objeto (estructura más compleja)	{}



Tipos de datos

Básicos

</Riwi>

Data Types

Numbers

2, -3, 22.956

Important for calculations and code where you need to “work with a number”

Strings (Text)

'Hi', "Hi", `Hi`

Important for outputting results, gathering input

Booleans

true / false

Important for conditional code and situations where you only have 2 options

Objects

{ name: 'Max',
age: 31 }

Important for grouped/ related data, helps you with organizing data

Arrays

[1, 3, 5]

Important for list data, unknown amounts of data

Tipos de Datos

```
var text = "Hola, me llamo Manz";  
var number = 42;  
var boolean = true;  
var notDefined; // undefined
```

Typeof()

```
console.log(typeof text); // "String"  
console.log(typeof number); // "Number"  
console.log(typeof boolean); // "Boolean"  
console.log(typeof notDefined); // undefined
```

Tipos de Datos

Variables & Constants

```
let userName = 'Max';
```

A “data container” / “data storage”


```
userName = 'Manu';
```

...where the value can change!

```
const totalUsers = 15;
```

A “data container” / “data storage”

```
totalUsers = 20;
```



...where the value must not change!



Use **constants as often as possible** (i.e. whenever you actually got data that never changes) to be clear about your intentions (in your code).

Tipos de Datos

Variable Naming

Allowed

```
let userName
```

Best Practice:
camelCase

```
let ageGroup5
```

Only letters
and digits

```
let $kindOfSpecial
```

Starting with \$
is allowed

```
let _internalValue
```

Starting with _
is allowed

Not Allowed / Not Recommended

```
let user_name
```

Allowed but
bad practice!

```
let 21Players
```

Starting digits
not allowed

```
let user-b
```

No special
characters!

```
let let
```

Keywords not
allowed

Tipos de Datos

Constructor.name

```
console.log(text.constructor.name);      // String
console.log(number.constructor.name);    // Number
console.log(boolean.constructor.name);   // Boolean
console.log(notDefined.constructor.name);
// ERROR, sólo funciona con variables definidas
```



Tipos de funciones

Propiedades

Constructor	Descripción
<code>function nombre(p1, p2...) {}</code>	Crea una función mediante declaración .
<code>var nombre = function(p1, p2...) {}</code>	Crea una función mediante expresión .
<code>new Function(p1, p2..., code);</code>	Crea una función mediante un constructor de objeto .

Funciones

Funciones por declaración

Comment Code |

```
function saludar() {  
    return "Hola";  
}
```

```
saludar(); // 'Hola'
```

```
typeof saludar; // 'function'
```

Funciones

Funciones por expresión

```
// El segundo "saludar" (nombre de la función) se suele omitir:  
//es redundante  
const saludo = function saludar() {  
    return "Hola";  
};  
  
saludo(); // 'Hola'
```

Funciones

Funciones como objetos

```
const saludar = new Function("return 'Hola'");  
  
saludar(); // 'Hola'
```

Funciones anónimas

```
// Función anónima "saludo"  
const saludo = function () {  
    return "Hola";  
};  
  
saludo; // f () { return 'Hola'; }  
saludo(); // 'Hola'
```



“

Ahora que conocemos las **funciones anónimas**, podremos comprender más fácilmente como utilizar **callbacks** (también llamadas funciones callback o retrollamadas). A grandes rasgos, un **callback** (llamada hacia atrás) es pasar una **función B por parámetro** a una **función A**, de modo que la función A puede ejecutar esa función B de forma genérica desde su código, y nosotros podemos definirlas desde fuera de dicha función

”

Funciones

Callbacks

```
// fB = Función B
const fB = function () {
  console.log("Función B ejecutada.");
};

// fA = Función A
const fA = function (callback) {
  callback();
};

fA(fB);
```

```
// fB = Función B (callback)
const fB = function () {
  console.log("Función B ejecutada.");
};

// fError = Función Error (callback)
const fError = function () {
  console.error("Error");
};

// fA = Función A
const fA = function (callback, callbackError) {
  const n = ~~(Math.random() * 5);
  if (n > 2) callback();
  else callbackError();
};

fA(fB, fError); // Si ejecutamos varias veces, algunas
//darán error y otras no
```

“

podemos planear ejecutar la función `FA()` cambiando los callbacks según nos interese, sin necesidad de crear funciones con el mismo código repetido una y otra vez. Además, en el caso de que las funciones callbacks sean muy cortas, muchas veces utilizamos directamente la función anónima, sin necesidad de guardarla en una variable previamente

”

Funciones

Callbacks

```
// fA = Función A
const fA = function (callback, callbackError) {
  const n = ~~(Math.random() * 5);
  if (n > 2) callback();
  else callbackError();
};

fA(
  Comment Code
  function () {
    console.log("Función B ejecutada.");
  },
  Comment Code
  function () {
    console.error("Error");
  }
);
```

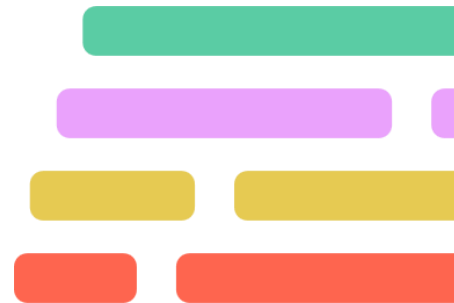


Funciones

Funciones autoejecutables

```
// Función autoejecutable
(function () {
    console.log("Hola!!");
})();

// Función autoejecutable con parámetros
(function (name) {
    console.log(`¡Hola, ${name}!`);
})("Manz");
```



Funciones

Funciones autoejecutables

Ten en cuenta, que si la función autoejecutable devuelve algún valor con return, a diferencia de las funciones por expresión, en este caso lo que se almacena en la variable es el valor que devuelve la función autoejecutada.

```
const f = (function (name) {  
  return `¡Hola, ${name}!`;  
})("Manz");  
  
f; // '¡Hola, Manz!'  
typeof f; // 'string'
```



Funciones

Clausuras

Las **clausuras** o cierres, es un concepto relacionado con las funciones y los ámbitos que suele costar comprender cuando se empieza en Javascript. Es importante tener las bases de funciones claras hasta este punto, lo que permitirá entender las bases de una clausura.

```
// Clausura: Función incr()
const incr = (function () {
  let num = 0;
  return function () {
    num++;
    return num;
  };
})();

typeof incr; // 'function'
incr(); // 1
incr(); // 2
incr(); // 3
```

Tenemos una función anónima que es también una función autoejecutable. Aunque parece una función por expresión, no lo es, ya que la variable **incr** está guardando lo que devuelve la función anónima autoejecutable, que a su vez, es otra función diferente.

Operadores Básicos

Aritméticos

Nombre	Operador	Descripción
Suma	$a + b$	Suma el valor de a al valor de b.
Resta	$a - b$	Resta el valor de b al valor de a.
Multiplicación	$a * b$	Multiplica el valor de a por el valor de b.
División	a / b	Divide el valor de a entre el valor de b.
Módulo	$a \% b$	Devuelve el resto de la división de a entre b.
Exponenciación	$a ** b$	Eleva a a la potencia de b, es decir, a^b . Equivalente a <code>Math.pow(a, b)</code> .



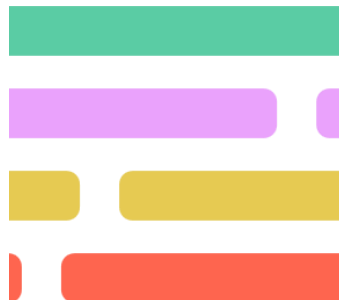
Operadores Básicos

Aritméticos

Operators

+	Add two numbers
-	Subtract two numbers
*	Multiply two numbers
/	Divide two numbers
%	Divide two numbers, yield remainder
**	Exponentiation (e.g. $2 ** 3 = 8$)

=	Assign value to variable
---	--------------------------



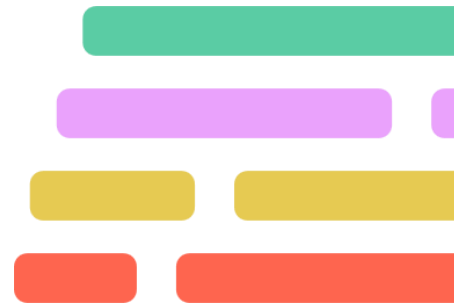
Operadores Básicos

Operador módulo

```
const numbers = [10, 20, 30, 40, 50, 60, 70, 80];

for (let i = 0; i < numbers.length; i++) {
  const mod = i % 2;
  console.log(numbers[i], numbers[mod]);
}
```

```
10 10
20 20
30 10
40 20
50 10
60 20
70 10
80 20
```



Operadores básicos

Operador de exponenciación

</Riwi>

```
const a = 2;  
const b = 5;  
  
console.log(Math.pow(a, b));    // 32  
console.log(a ** b);           // 32  
console.log(a * a * a * a * a); // 32
```



Operadores Básicos

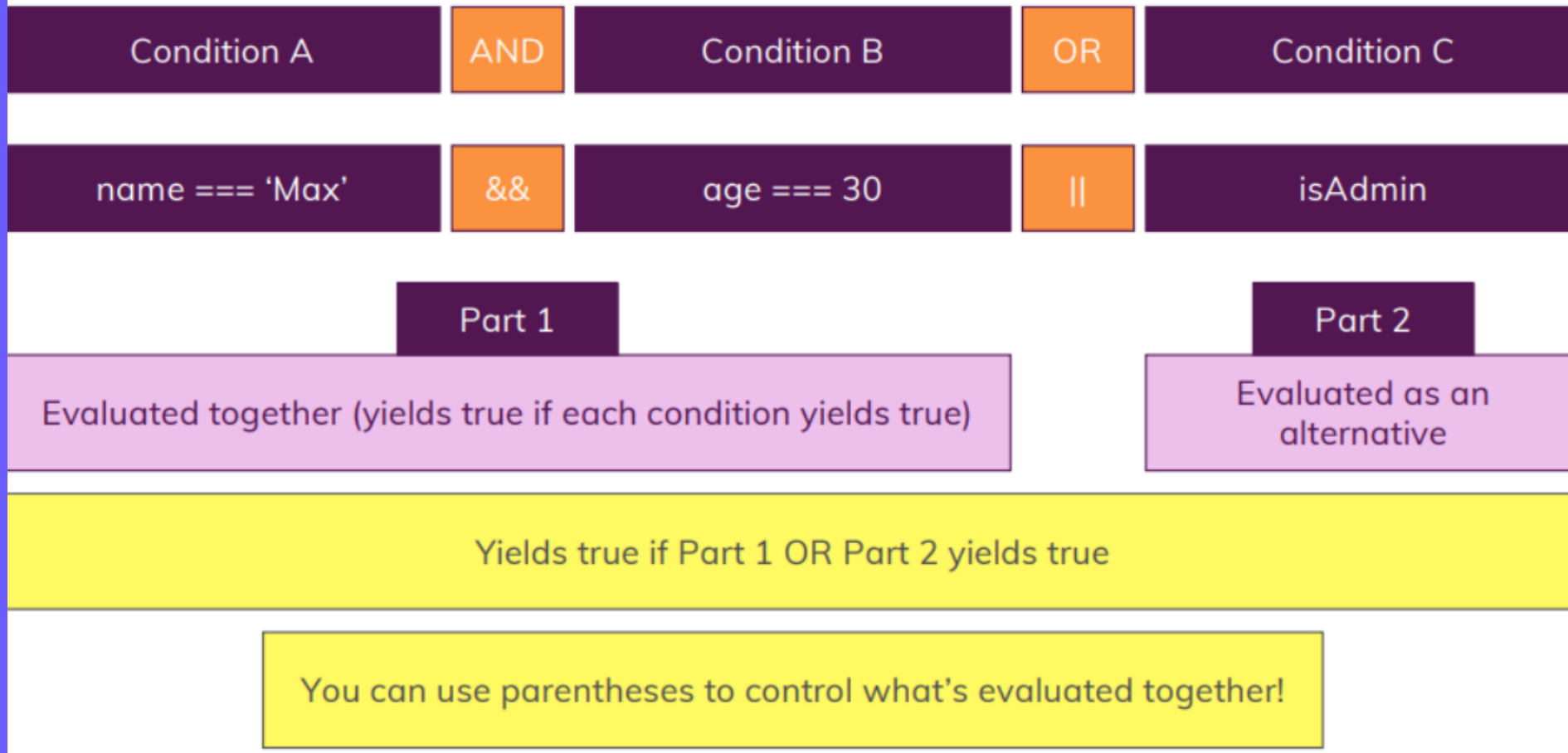
Operadores de asignación

Nombre	Operador	Descripción
Asignación	$c = a + b$	Asigna el valor de la parte derecha (en este ejemplo, una suma) a c.
Suma y asignación	$a += b$	Es equivalente a $a = a + b$.
Resta y asignación	$a -= b$	Es equivalente a $a = a - b$.
Multiplicación y asignación	$a *= b$	Es equivalente a $a = a * b$.
División y asignación	$a /= b$	Es equivalente a $a = a / b$.
Módulo y asignación	$a \% = b$	Es equivalente a $a = a \% b$.
Exponenciación y asignación	$a ** = b$	Es equivalente a $a = a ** b$.

Operadores Básicos

Operadores de asignación

Combining Conditions



Operadores Básicos

Operadores de asignación

Boolean Operators

Important For Conditional Code: Return **true** or **false**

==	Check for value equality	e.g. <code>a == b</code>
!=	Check for value inequality	e.g. <code>a != b</code>
=== and !==	Check for value AND type (in)equality	e.g. <code>a === b</code> / <code>a !== b</code>
> & <	Check for value being greater / smaller	e.g. <code>a > b</code> / <code>a < b</code>
>= & <=	Check for value being greater or equal / smaller or equal	e.g. <code>a >= b</code> / <code>a <= b</code>
!	Check if NOT true	e.g. <code>!a</code>

Prefer over ==

Operadores Básicos

Operadores de asignación

Beware of Objects & Arrays in Comparisons!

{ name: 'Max' }

=== or ==

{ name: 'Max' }

false!

Objects and arrays are kind of special in JavaScript!

Operadores Básicos

Operadores Unarios

</Riwi>

Nombre	Operador	Descripción
Incremento	a++	Usa el valor de a y luego lo incrementa. También llamado postincremento .
Decremento	a--	Usa el valor de a y luego lo decrementa. También llamado postdecremento .
Incremento previo	++a	Incrementa el valor de a y luego lo usa. También llamado preincremento .
Decremento previo	--a	Decrementa el valor de a y luego lo usa. También llamado predecremento .
Resta unaria	-a	Cambia de signo (niega) a a.

Operadores Básicos

Operadores de comparación

Nombre	Operador	Descripción
Operador de igualdad ==	a == b	Comprueba si el valor de a es igual al de b. No comprueba tipo de dato.
Operador de desigualdad !=	a != b	Comprueba si el valor de a no es igual al de b. No comprueba tipo de dato.
Operador mayor que >	a > b	Comprueba si el valor de a es mayor que el de b.
Operador mayor/igual que >=	a >= b	Comprueba si el valor de a es mayor o igual que el de b.
Operador menor que <	a < b	Comprueba si el valor de a es menor que el de b.
Operador menor/igual que <=	a <= b	Comprueba si el valor de a es menor o igual que el de b.
Operador de identidad ===	a === b	Comprueba si el valor y el tipo de dato de a es igual al de b.
Operador no idéntico !==	a !== b	Comprueba si el valor y el tipo de dato de a no es igual al de b.

Operadores Básicos

Operadores binarios

</Riwi>

Nombre	Operador	Descripción
Operador AND	$a \& b$	Devuelve 1 si ambos operandos son 1.
Operador OR	$a b$	Devuelve 1 si al menos un operando es 1.
Operador XOR (OR exclusivo)	$a \wedge b$	Devuelve 1 si ambos operandos son diferentes.
Operador NOT (unario)	$\sim a$	Invierte los bits del operando (por ejemplo, 000101 pasa a 111010). Trunca a 32 bits.
Operador LEFT SHIFT	$a \ll b$	Desplazamiento de bits hacia la izquierda. Ej: 11 (3) pasa a 110 (6).
Operador RIGHT SHIFT	$a \gg b$	Desplazamiento de bits hacia la derecha. Ej: 11 (3) pasa a 1 (1).
Operador RIGHT SHIFT sin signo	$a \ggg b$	Desplazamiento de bits hacia la derecha, como un operador sin signo.

Operadores Básicos

Asignación a nivel de bit

Nombre	Operador	Descripción
Desplazamiento a la izquierda y asignación	<code>a <<= b</code>	Es equivalente a <code>a = a << b</code> .
Desplazamiento a la derecha y asignación	<code>a >>= b</code>	Es equivalente a <code>a = a >> b</code> .
Desplazamiento a la derecha sin signo y asignación	<code>a >>>= b</code>	Es equivalente a <code>a = a >>> b</code> .
Operación AND y asignación	<code>a &= b</code>	Es equivalente a <code>a = a & b</code>
Operación OR y asignación	<code>a = b</code>	Es equivalente a <code>a = a b</code>
Operación XOR y asignación	<code>a ^= b</code>	Es equivalente a <code>a = a ^ b</code>
Operación AND lógico y asignación	<code>a &&= b</code>	Es equivalente a <code>a && (a = b)</code>
Operación OR lógico y asignación	<code>a = b</code>	Es equivalente a <code>a (a = b)</code>

Operadores avanzados

Operadores String y concatenación

Nombre	Operador	Descripción
Concatenación de texto	a + b	Une el contenido de a con el contenido de b
Conversión a número (Suma unaria)	+a	Si a no es un número, intenta convertirlo en un número.

Ejemplo	Resultado	Explicación
-----	-----	-----
2 + 2	// 4 (Número + número)	2 + 2
"2" + "2"	// "22" (String + string)	String(2) + String(2)
"2" + 2	// "22" (String + número)	String(2) + 2
2 + "2"	// "22" (Número + string)	2 + String(2)
"a" + 2	// "a2" (String + número)	String("a") + 2

Operadores avanzados

Operadores Logicos

Nombre	Operador	Descripción
Operador lógico AND	a && b	Devuelve a si es false, sino devuelve b.
Operador ternario ?:	a ? b : c	Si a es true, devuelve b, sino devuelve c.
Operador lógico OR	a b	Devuelve a si es true, sino devuelve b.
Operador lógico Nullish coalescing	a ?? b	Devuelve b si a es null o undefined, sino devuelve a.
Operador de asignación lógica nula ??=	a ??= b	Es equivalente a a ?? (a = b)
Operador de encadenamiento opcional ?.	data?.name	Permite intentar acceder a una propiedad, aunque su padre no exista.
Operador unario lógico NOT	!a	Invierte el valor. Si es true devuelve false y viceversa.

**</Be a
coder>**