



# Curso básico de Javascript

🕒 Created	@January 18, 2022 10:01 AM
📎 Materials	
☰ Curso	
▼ Status	Done
🌐 URL	<a href="https://platzi.com/clases/basico-javascript/">https://platzi.com/clases/basico-javascript/</a>

- Genera dinamismo en páginas web y permite interacción entre usuarios.
- JS es un lenguaje **interpretado** (traduce la manera como escribimos las personas a la manera como funciona un computador), **orientado a objetos** (trabaja con objetos que encapsulan variables y métodos para interactuar con otros objetos), **débil/ tipado** (permite operaciones entre datos de diferentes tipos. ej: strings e integers) y **dinámico** (no requiere compilación previa.... permite probar código inmediata):
- JS tiene una comunidad muy grande de desarrolladores y muchas librerías:
  - Dillo. web: Angular, VUE y React
  - App nativas (android / ios): React native
  - App escritorio (mac / windows): Electron
  - Backend + IoT: Node
- De manera alterna, la W3 avala la opción de "**webassembly**", la cual permite hacer todo en un mismo lugar en vez de tener que combinar HTML, CSS y JS.

## Valores

### ▼ Valores básicos:

- **integers**: números enteros
- **strings**: textos "..."
- **boolean**: true o false
- **null**: valores vacíos
- **undefined**: valores que no reconoce pero que posterior/ podría hacerlo

### ▼ Valores tipo objeto:

- **array**: [1, 2, 3]
- **objeto**: {nombre: "Alejo", edad: 38, ciudad: Medellín}. Equivalente a un diccionario en Python según interpreto

Con **typeof** puedo averiguar el tipo de valor de un dato.

Con **//** puedo hacer comentarios al código

## Variables

Variables son un "contenedor" q almacena datos

```
var xyz = "Alejo"
```

Tiene 2 estados:

- Declarar: reserva palabra en memoria (**var xyz**)
- Definir: asigna valor a palabra reservada (**= "Alejo"**)
- Inicializar: cuando previamente se reservó en memoria y se le va a asignar valor (**xyz = "Alejo"**)

## Funciones

Conjunto de sentencias para generar ciertas acciones con valores almacenados en variables.

Existen 2 tipos de fxnes:

- **Declarativa**: reserva automática/ con un nombre. Permite ser llamada antes de ser declarada.

```
function miFxn () {  
  return 3;  
}
```

- **Expresiva**: variable que almacena una fxn.

```
var miFxn = function (a,b) {  
  return a + b  
}
```


Para ensayar / correr código puedo usar la consola del navegador (ej: chrome). En Mac: **Cmd + Alt + I**

**console.log** es una fxn q me permite imprimir cosas en la consola del navegador.

Link útil para profundizar:

### Functions - JavaScript | MDN

Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the

 <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions>



MDN Web Docs  
moz://a

## Scope

Alcance que tienen las variables. Condiciona variables a las que se puede acceder.

Existen 2 tipos de alcance:

- **Global**: es como el "universo". Variables son accesibles desde cualquier parte del código. Comienza cada vez q se abre el navegador.
- **Local**: es como los "planetas". Variables son accesibles solo cuando se está dentro de ese planeta... no son visibles para otros planetas. Comienza cada vez que se define una fxn.

## Hoisting

Cuando las variables y fxnes se declaran antes q se procese cualquier tipo de código.

⚠ es diferente a "hosting"

Una **buena práctica** consiste en **declarar** todas las fxnes q se van a usar al inicio del código. Esto evita problemas con fxnes q usen variables q no han sido declaradas.

## Coerción

Es la manera para cambiar el tipo de un valor (ej: número a string o viceversa). Existen coerciones **implícitas** (JS cambia de un tipo a otro de manera automática según sus reglas) y **explícitas** (el tipo cambia según lo q yo deseo).

Útil para hacer operaciones matemáticas con input's de usuario en una página xej, las cuales JS las interpreta como string inicial/.

## Truthy y Falsy

Son valores que son verdaderos o falsos, los cuales **condicionan** acciones en el código

### ▼ Falsy (False)

- "" // un string vacío
- 0 -0
- null
- NaN
- false
- undefined

### ▼ Truthy (True)

- "hola"
- 30
- {a:1}
- [1,3]
- true
- function a(){}
- más

## Expresiones y operadores

Operadores: nos permiten hacer cosas con los datos. Ejemplo (+, -, \*, /) q son operadores **binarios** los cuales nos permiten hacer cosas entre dos números. Pero también fxa con string's xej para hacer concatenaciones.

Los operadores **unitarios** trabajan con un solo valor. Nos permiten xej negar: **!false**

- Operador de asignación: `a = 1`
- Operador de comparación de valor: `3 == "3"` (resultado: true)
- Operador de comparación de valor y tipo: `3 === "3"` (resultado: false)
- Operador mayor, menor, etc (`<`, `>`, `<=`, `>=`)
- Operador **"and"** donde ambas variables deben ser verdadera: `a && b`
- Operador **"or"** donde una de dos variables debe ser verdadera: `a || b` (en Mac se obtiene con **Alt + 1**)
- Operador para incrementar el valor de un número en una unidad: `++`

```
var edad = 38
edad++

edad = 39
```

Operador para incrementar el valor de un número en una cantidad específica: `+=`

```
var edad = 38
edad += 4

edad = 42
```

#### Expressions and operators - JavaScript | MDN

This chapter describes JavaScript's expressions and operators, including assignment, comparison, arithmetic, bitwise, logical, string, ternary and more. A complete and detailed list of operators and expressions is also available in the reference. JavaScript has the following types of operators. This section describes the operators

 [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions\\_and\\_Operators](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators)



MDN Web Docs  
moz://a

## Condicionales

**if / else**: evalúan dos condiciones... si true o false (ej: si algo es igual o mayor o menor... etc, haga una cosa en especial)


**else if**: permite evaluar varias condiciones al mismo tiempo (más de 2).

```
if (jugador == "papel" && computador == "piedra") {
  console.log(`ganaste con ${jugador} sobre ${computador}`);
} else if (jugador == "piedra" && computador == "tijera") {
  console.log(`ganaste con ${jugador} sobre ${computador}`);
} else if (jugador == "tijera" && computador == "papel") {
  console.log(`ganaste con ${jugador} sobre ${computador}`);
} else {
  console.log(`perdiste con ${jugador} contra ${computador}`);
}
```

Operador ternario: evalúa en una sola línea de código (una especie de "if y else" en una sola línea). Si se cumple la condición (true) entonces hace la primera cosa que le digo; pero si no se cumple (false), hace la segunda.

#### Operador condicional (ternario) - JavaScript | MDN

Si la condición es true, el operador retorna el valor de la expr1; de lo contrario, devuelve el valor de expr2. Por ejemplo, para mostrar un mensaje diferente en función del valor de la variable isMember, se puede usar esta declaración:

 [https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional\\_Operator](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional_Operator)



MDN Web Docs

moz://a

Switch: También permite validar una condición. Hace lo mismo que el **if** pero su sintaxis es diferente. Evalúa casos específicos con un operador **===**. Si hay varios casos, se debe usar **break** para que la validación se rompa y devuelva el valor que espero de la misma (que no imprima todos los resultados posibles)

```
switch (true) {
  case (jugador == "papel" && computador == "piedra"):
    console.log(`ganaste con ${jugador} sobre ${computador}`);
    break;
  case (jugador == "piedra" && computador == "tijera"):
    console.log(`ganaste con ${jugador} sobre ${computador}`);
    break;
  case (jugador == "tijera" && computador == "papel"):
    console.log(`ganaste con ${jugador} sobre ${computador}`);
    break;
  default:
    console.log(`perdiste con ${jugador} contra ${computador}`);
}
```

## Array

Valor tipo objeto porque almacena datos en su interior [...]. Elementos deben ir separados por comas (strings, #s, arrays, etc).

**length** me permite saber cantidad de elementos al interior de un array:

```
console.log (nombre-array.length)
```

Para acceder a un elemento determinado del array a través del índice, uso corchetes para poner el número. Recordar que en programación, la cuenta inicia con el 0... es el primer elemento (no el 1):

```
console.log (nombre-array[2])
```

Los arrays pueden mutar usando algunas fxnes:

- **push**: añade elementos al final de un array previamente creado.

```
var datos = ["Alejo", "Antonio", "Naranja"]

var masdatos = datos.push ("Gaviria");
```

```
["Alejo", "Antonio",  
"Naranjo", "Gaviria"]
```

- **pop**: elimina el último elemento de un array

```
var eliminando = datos.pop('Antonio');  
// Aunque indique el nombre de un elemento  
// se elimina es el último elemento  
  
["Alejo", "Antonio",  
"Naranjo"]
```

- **unshift**: agrega valores al inicio del array

```
var elementoinicio = datos.unshift("Picachú")  
  
['Picachú', 'Alejo',  
'Antonio', 'Naranjo',  
'Gaviria']
```

- **shift**: elimina elemento q está al inicio

```
var borrarinicio = datos.shift("Picachú")  
  
['Alejo', 'Antonio',  
'Naranjo', 'Gaviria']
```

- **indexOf**: saber cuál es la posición (index) de un elemento del array

```
var posicion = datos.indexOf("Naranjo")  
  
posicion  
2
```

## Loops

Loops o "ciclos". Permite repetir una tarea de manera automática, mientras se cumplan determinadas condiciones.

Contexto previo para usar código:

```
var estudiantes = [ 'Maria', 'Sergio', 'Rosa', 'Daniel' ];  
  
function saludarEstudiante(estudiante) {  
    console.log(`Hola, ${estudiante}`);  
}
```

- Opción con variable "i":

```
for (var i = 0; i < estudiantes.length; i++) {
  // cree una variable "i" q inicia con valor 0
  // Mientras "i" sea menor q la longitud del array, es válido el loop
  // en cada iteración modifique el valor de "i" aumentando una unidad
  saludarEstudiante(estudiantes[i]);
}
```

- Opción "of"... el singular de un array

```
for (var estudiante of estudiantes) {
  saludarEstudiante(estudiante);
}
```

- Opción "while"

```
while (estudiantes.length > 0) { // Aquí la tarea se hará siempre y cuando sea true, cuando llegué a false, dejará de hacer la tarea
  var estudiante = estudiantes.shift(); // shift() es un método que saca un elemento del array de la posición 0 a la última, Pop() c
  saludarEstudiante(estudiante);
}
```

## Objects

Un objeto es una colección de propiedades, y una propiedad es una asociación de key (nombre, o clave) y valores... como el diccionario en Python. Se usan {clave: valor}.

Múltiples propiedades se separan con comas.

```
var misDatos = {
  nombre: "Alejandro", // key - value
  apellido: "Naranjo",
  annio: 1983,
  detallesMios: function() { // Metodo de un objeto (una función dentro de un objeto)
    return `Datos de ${this.nombre} ${this.apellido} que nació en ${this.annio}`;
  }
};
```

Para obtener un valor determinado, debo indicar su clave. Ej: `misDatos.apellido`. Si es una fxn al interior: `misDatos.detallesMios()`.

### Fxn constructora

Si quiero generar objetos de una manera más automatizada.

```
function persona(nombre, apellido, annio){
  this.nombre = nombre;
  this.apellido = apellido;
  this.annio = annio;
}

// Aquí empiezo a usar la fxn creada:
var persona1 = new persona("Alejandro", "Naranjo", 1983)
var persona2 = new persona("Javier", "Naranjo", 1930)
var persona3 = new persona("Gloria", "Gaviria", 1942)
```

## Recorrer Array's

Para trabajar con array's cuyo contenido sea un objeto

- **filter**: filtra cosas validando si algo es verdadero o falso, metiéndolo en un nuevo array (no modifica array original).

```
//Base para el ejercicio (crear array con objetos):

var articulos = [
  { nombre: 'Bici', costo: 3000 },
  { nombre: 'TV', costo: 2500 },
  { nombre: 'Libro', costo: 320 },
  { nombre: 'Celular', costo: 10000 },
  { nombre: 'laptop', costo: 20000 },
  { nombre: 'teclado', costo: 500 },
  { nombre: 'audifonos', costo: 1700 }
]

// Aplicando "filter":
var articulosFiltrados = articulos.filter(function(articulo){
  return articulo.costo <= 500 /* Menor o igual a 500 */
});

console.log(articulosFiltrados);
```

- **map**: mapea (encontrar) artículos y también requiere generar nuevo array.

```
var nombreArticulos = articulos.map(function(articulo){
  return articulo.nombre
});

console.log(nombreArticulos);
```

- **find**: ayuda a encontrar algo dentro de un array. Si existe, lo genera; sino, no sucede nada.

```
var encuentraArticulo = articulos.find(function(articulo){
  return articulo.nombre === "laptop"
});
```

- **forEach**: no genera nuevo array sino q filtra un array q ya existe.

```
articulos.forEach(function(articulo){
  console.log(articulo.nombre);
});
```

- **some**: regresa una validación de verdadero o falso si se cumple o no la condición.

```
var articulosBaratos = articulos.some(function(articulo){
  return articulo.costo <= 700;
});
```

- **f**: ada.



