



JavaScript

Script Programming Language



Conceptos JavaScript

¿Qué es JavaScript?

Es un lenguaje de programación interpretado, que admite construcción de objetos basada en prototipos, contenidos de actualización dinámica, agregar mayor interactividad a la web y ayudan a brindar una mejor experiencia al usuario. La mayoría de los sitios web la emplean el lenguaje y es compatible con todos los navegadores modernos, su poder se refleja principalmente en el frontend.

Puede ser utilizado con las librerías y framework como: angular, jquery, backbone, react y demás, escritas sobre **JavaScript**.



JavaScript


Características

Interpretado

Orientado a objetos

Débilmente Tipado

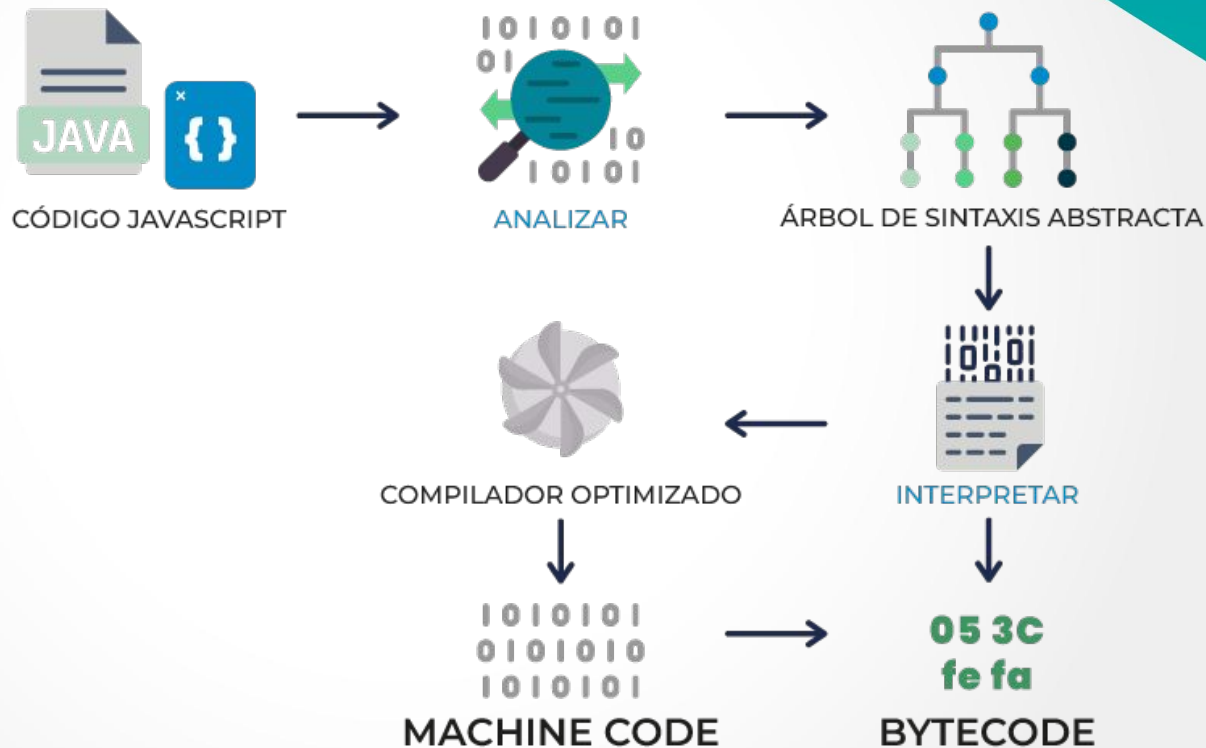
Dinámico



```
4 + "7"; //Esto es igual a '47'  
4 * "7"; // Esto es igual a 28  
4 * true; // Esto es igual a 4  
false - 3; //Esto es igual a -3
```



¿JavaScript es realmente un lenguaje interpretado?





Backward & Forward

Backward

En cambio, **JavaScript** si es un backward compatible, esto quiere decir que todas las funciones nuevas no dañará el trabajo ya hecho, pero no se podrán utilizar en nuestro entorno de trabajo inmediatamente hasta que la W3C lo estandarice.

Para solucionar este problema existe **BABEL** que es un compilador que permite utilizar estas nuevas tecnologías de JavaScript traduciendo la nueva versión de javascript a una versión que el navegador pueda entender.

BABEL



Backward & Forward

Forward

Ser compatible con versiones futuras significa que incluir una adición al lenguaje en un programa no causaría que el código se rompa y deje de funcionar si se ejecuta en un motor anterior.

Javascript NO es un lenguaje compatible con versiones futuras.



¿Por qué JavaScript?

Hasta el 2019, JavaScript era parte de 3 estándares que puedes utilizar para construir productos web: HTML, CSS y JS.

Comunidad

Frontend

Backend

Apps Móviles

Aplicaciones para Escritorio



Elementos de un Lenguaje de Programación



Elementos de un Lenguaje de Programación





Variables

Una variable es la representación del espacio en memoria que vamos a reservar para poder guardar un valor, y ese valor puede ser de diferentes tipos.

Cómo declarar una variable

```
var marca_carro = "Ford";
```

Dato en consola:

```
> marca_carro;  
> "Ford"
```



Variables

Hay dos estados para una variable

El primero es
declarar la variable

```
//Declarar  
var edad;
```

Inicializar la
variable

```
//Declarar  
var edad;  
//Inicializar  
edad = 18;
```

Variables complejas

```
var usuario = {  
  nombre_usuario: "arochaking", //Variable tipo  
  objeto  
  contraseña: "Hola123"  
}
```

```
var colores = ["Rojo", "Verde", "Azul"]; //Variable  
tipo Array
```

```
> usuario;  
< {nombre_usuario: "arochaking", contraseña:  
  "Hola123"}
```

```
> colores;  
< (3) ["Rojo", "Verde", "Azul"]
```



Funciones

Las funciones son un conjunto de sentencias o instrucciones que podemos utilizar para generar unas acciones o tareas con los datos que guardamos en las variables.

¿Qué son las funciones en Javascript

JavaScript



Es un conjunto de instrucciones que realiza una tarea o calcula un valor



Una función es un trozo de código reutilizable en el que hay un conjunto de instrucciones, este código solo se efectura cuando se llama a dicha función.



DECLARACIÓN

1 DECLARAR

- * Palabra reservada `function`
- * Nombre de la función
- * Escribir los Parámetros

2 BLOQUE DE CÓDIGO

- * Se escribe el conjunto de instrucciones que va a realizar la función

3 INVOCAR FUNCIÓN

- * Llamar a la función
- * Escribir los argumentos (valores)

```
// Function declaration //  
function someName(param1,param2) {  
    // bunch of code as needed...  
    var a = param1 + "love" + param2;  
    return a;  
}  
  
// Invoke (run / call) a function  
someName("Me","You")
```

1

2

3



```
//Declarativas
function saludar(nombre) {
  console.log(`Hola ${nombre}`);
}

saludar("Diego");

//Otro ejemplo

function sumar(a,b) {
  var resultado = a + b;
  return resultado;
}

sumar(1,3); //En la consola aparecería el resultado
es decir 4
```

Tipo de Funciones Declarativas

Hay dos tipos de funciones: **declarativas** y de **expresión**.

NOTA: Cada parámetro va separado por una coma. Cada instrucción que tenga la función debe terminar con punto y coma.



Tipos de Funciones

De expresión o anónimas

```
//De expresión
var saludar = function(nombre) {
  console.log(`Hola ${nombre}`)
}

saludar('Diego');
```

En la **expresión** de función, la función podría o no llevar nombre, aunque es más común que se hagan anónimas.

Diferencias

A las funciones declarativas se les aplica hoisting, y a la expresión de función, no. Ya que el hoisting solo se aplica en las palabras reservadas var y function.

	Declarativa	Anónimas
Hoisting	SI	NO



Parámetros de las Funciones

Los parámetros son aquellos valores que van dentro de paréntesis al colocar una función, estos son los datos que la función tomará en cuenta para ejecutar la instrucción.

```
function saludo(nombre, apellido){  
    alert("Hola " + nombre + " " + apellido);  
}  
  
saludo("Sebastian", "Arocha");  
  
//Otro ejemplo  
  
function miSuma(a,b){  
    return a + b;  
}  
  
miSuma(1,2); //En la consola diría 3
```



Bases de JavaScript

Scope

Scope

El scope **decide a qué variables tienes acceso** en cada parte del código.

Local

```
function nombre(){  
  var miNombre = "Sebastian";  
  console.log("Hola " + miNombre);  
}
```

Global

```
var miApellido = "Arocha";  
  
function saludo(){  
  var miNombre = "Sebastian";  
  console.log("Hola " + miNombre + " " +  
    miApellido);  
}  
  
//la variable local puede acceder a la variable  
global mas no al revés
```



Hoisting

El Hoisting es un **error** que ocurre cuando las variables y las funciones se declaran antes de que se procese cualquier tipo de código.

```
saludo(); //Se declara la variable antes de  
asignarle un valor
```

```
function saludo() {  
    console.log("Hola " + nombre);  
}
```

```
var nombre = "Sebastian"; //se le asigna un  
valor a la variable despues que se ejecuta
```

OUTPUT

Hola undefined



Hoisting

Pasos del compilador

```
saludo();  
  
function saludo() {  
    console.log("Hola " + nombre);  
}  
  
var nombre = "Sebastian";
```



```
var nombre;  
  
function saludo() {  
    console.log("Hola " + nombre);  
}  
  
saludo();  
  
var nombre = "Sebastian";
```



Hoisting

Pasos del compilador

```
var nombre =undefined;
```

```
function saludo() {  
    console.log("Hola " + nombre);  
}
```

```
saludo();
```

```
nombre = "Sebastian";
```

```
saludo()
```



Hola undefined



Bases de JavaScript

Coerción

Tipos de coerción

Coerción implícita = es cuando el lenguaje nos ayuda a cambiar el tipo de valor.

Veamos un ejemplo:

```
var a = 4 + "7"; //el signo de + es una concatenación y convierte el 4 en string
```

Nos **"ayuda"** convirtiendo automáticamente la variable en string.

```
var b = 4 * "7";  
//convierte el string a número
```

JavaScript le da **prioridad al número** y convierte el string en un valor numérico.

```
type of a;  
> string
```

```
a  
> 47
```

El lenguaje convierte **el número a un string**, y representa una concatenación.

Efectúa la operación matemática y convierte el "7" string a número.

```
type of b;  
> number
```

```
b  
> 28
```



Bases de JavaScript

Coerción

Coerción explícita = es cuando obligamos a que cambie el tipo de valor. Ejemplo:

```
var a = 20;  
var b = a + ""; //la concatenamos para que a se convierta en un string
```

← Cuando tenemos una variable número y queremos convertirla en string

```
typeof b;  
> string
```

```
b  
> 20
```

Se convirtió en string

```
typeof c;  
> string
```

```
c  
> 20
```

En la consola veríamos lo siguiente

```
var c = String(a)
```

```
typeof d;  
> number
```

```
d  
> 20
```

Automáticamente convierte todo lo que tenemos en la variable string en número.

```
var d = Number(c)
```

Para convertirlo en número usamos el dato **Number**



Bases de JavaScript

Valores: Truthy and Falsy

¿Cuáles son los valores que por defecto son verdadero o falso?

```
//Ejemplos en los que Boolean devuelve False:
```

```
Boolean(0); //false  
Boolean(null); //false  
Boolean(NaN); //false  
Boolean(undefined); //false  
Boolean(false); //false  
Boolean(""); //false
```

```
//Ejemplos en los que Boolean devuelve True:
```

```
Boolean(1); //true para 1 o cualquier número diferente de cero (0)  
Boolean("a"); //true para cualquier caracter o espacio en blanco en el string  
Boolean([]); //true aunque el array esté vacío  
Boolean({}); //true aunque el objeto esté vacío  
Boolean(function(){}); //Cualquier función es verdadera también
```




Bases de JavaScript

Operadores: Asignación, Comparación y Aritméticos

Estos operadores son bastante utilizados cuando trabajamos con condicionales, hay distintos tipos de ellos.

```
//Operadores binarios:
```

```
3 + 2 //Suma
```

```
50 - 10 // Resta
```

```
10 * 20 //Multiplicación
```

```
20 / 2 //División
```

```
2 ** 6 //Potenciación
```

```
"Sebastian " + "Arocha" //concatenación de strings
```

```
//Operadores unitarios:
```

```
!false //negativo de falso = true
```

```
//Operadores de asignación:
```

```
var a = 1; //Asignamos un valor a la variable
```



Bases de JavaScript

Operadores: Asignación, Comparación y Aritméticos

```
//Operadores de asignación:  
var a = 1; //Asignamos un valor a la variable  
  
//Operadores para comparar:  
3 == "3"; //Compara los valores y devuelve "true" en este caso  
  
3 === "3"; //Compara y valida los tipos y valores. Devuelve "falso" en este caso  
  
5 < 3 //Compara y valida si el 5 es menor a 3  
5 > 3 //Compara y valida si el 5 es mayor a 3  
5 <= 6 //Compara y valida si el 5 es menor o igual al 6  
5 >= 6 //Compara y valida si el 5 es mayor o igual al 6
```

```
a && b //Valida si ambas variables son verdaderas para que se cumpla la condición  
a || b //Aquí se cumple la condición si alguna de las dos variables es verdadera  
  
var edad = 40  
edad++ //Incrementa el valor en 1  
  
edad += 2 // Es lo mismo que decir edad = edad + 2 que es igual a 43  
//Igualmente este operador se pueden usar para las demas operaciones  
// Solo funciona cuando la variable tiene  
// un valor numerico asignado previamente
```



Condicionales

If, Else, Else if

Las condicionales son aquellas reglas que sirven para validar si algo es verdadero o falso y podemos generar ciertas acciones.

Condición If

```
//Sintaxis if
if (true) {
    console.log("Hola");
}
```

Condición Else

```
//Sintaxis Else
if (true) {
    console.log("Bienvenido");
}
else {
    console.log("Tu conexión con
esta página ha terminado");
}
```



Condicionales

If, Else, Else if

Las condicionales son aquellas reglas que sirven para validar si algo es verdadero o falso y podemos generar ciertas acciones.

Condición Else If

```
//Sintaxis if
if (true) {
    console.log("Hola");
}
```

Operador Ternario

```
condition ? true : false;

//condición-variable ?
verdadero : falso

var number = 1;

//verdadero //falso
var outcome = number === 1 ?
"I'm true" : "I'm false";
```

Condicionales: If, Else, Else if

Juego de Piedra, Papel o Tijera

Ejercicio Práctico



Reglas de Piedra, Papel y Tijera





Condicionales

Switch

Switch es una forma de generar una condición, donde podemos validar si algo es verdadero o falso, en switch tenemos los casos, y dependiendo si la condición de ese caso se cumple se ejecuta el bloque de código. Funciona distinto que if, pero hace lo mismo.

En el switch, los casos son condiciones que se tienen que validar. El uso de **break** es de suma importancia, ya que al no aplicarse se ejecutaría lo que está dentro de los siguientes casos sin validar si son verdaderos o no.

```
//Sintaxis de switch
var numero = 1;

switch (numero) {
  case numero = 1:
    console.log("Soy un uno!");
    break; //Si el caso es verdadero
se ejecuta y termina el switch
  case numero = 10:
    console.log("Soy un 10!");
    break; //Si el 1er caso es falso,
se valida si el segundo es verdadero
y termina el switch
  default: //Si ninguno de los dos
casos se cumplen
    console.log("Error");
}
```

Condicional: Switch

Juego de Piedra, Papel o Tijera

Ejercicio Práctico



Array

Índice y métodos

Los Arrays son una estructura de datos tipo objeto, es decir que van a guardar valores o un conjunto de datos.

```
//Sintaxis Array
var frutas = ["Manzana",
"Platanos", "Cereza", "Fresa"];
```

```
var jugadores = ["Messi", "Ronaldo",
"Beckham", "Suárez"];

jugadores.length //Nos mostrará la
longitud del array de cuantos datos tiene
> 4

//Si queremos acceder a los valores del
array según su posición
jugadores[0];
> "Messi"
jugadores[3]
> "Suárez"

var posicion =
jugadores.indexOf("Beckham");
//Nos sirve cuando queremos saber en que
posición está un elemento
> undefined
console.log(posicion);
> 2
```

Index del array



Array

Método del array

```
//Lista de invitados para una fiesta
var invitados = ["Karla", "Diego", "Luis", "Miriam"];
//Añadimos el array con los invitados
console.log(invitados);
> (4)

var añadirinvitados = invitados.push("Jose", "Luisa");
//Añade mas elementos a nuestro array
console.log(invitados);
> (6)

var quitarinvitados = invitados.pop("Luisa");
//Quita el último elemento de nuestro array
console.log(invitados);
> (5)

var nuevaLongitud = invitados.unshift("Carlos");
//Añade elementos al principio de nuestro array
console.log(invitados);
> (6) ["Carlos", "Karla", "Diego", "Luis", "Miriam", "Jose"]

var borrarPrimero = invitados.shift("Carlos");
//Elimina el primer elemento de nuestro array
console.log(invitados);
> (5) ["Karla", "Diego", "Luis", "Miriam", "Jose"]
```



Loops

For y For of

Los ciclos son una manera rápida y sencilla de repetir las instrucciones del código sin hacerlo de forma manual. Vamos a generar una función, y mientras dentro de esta función se cumpla la condición la tarea va a seguir repitiendo hasta que la condición deje de ser verdad.

Loops For

```
//Generamos un array
var paises = ["España", "Mexico", "Italia", "Francia"];

//Creamos la función o tarea que queremos que se ejecute
//El parámetro país hace referencia a que tendremos solo una variable
//en la función, podemos usar
//cualquier nombre pero debe tener relación con la función
function nombrarPaises(nombrepais) {
  console.log(`Bienvenido a los Juegos Olímpicos, ${nombrepais}`);
}

//Creamos el ciclo que repetirá esa tarea
//En este caso mientras i sea menor a 0, y luego i sea menor que la longitud
//de paises, i se sumará 1 cantidad
//Y luego se manda a llamar la función con el parámetro paises[i]
for (var i = 0; i < paises.length; i++) {
  nombrarPaises(paises[i]);
}
```



Loops

For y For of

La diferencia del **for of** con el **for** es que nombra cada elemento del array desde el primero al último, en vez del **for** que tiene una funcionalidad diferente depende del método que queramos.

Loops For Of

```
for (elemento of array) {  
  instrucción  
}  
  
//For of  
  
//Declaramos la variable con los elementos  
var alimentos = ["harina", "huevos", "azúcar", "avena"];  
  
//Escribimos la función que ejecutará el código  
function nombrarAlimentos(alimento) {  
  console.log(`Hay que comprar ${alimento}`);  
}  
  
//El ciclo ejecutará la función con cada elemento dentro del array  
for (var alimento of alimentos) {  
  nombrarAlimentos(alimento);  
}
```



While y do While

While

Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia.

//While - Sintaxis

```
while (condicion)
    sentencia
```

```
var numero = 1; //Creamos la variable
//Mientras la condicion sea verdad lo que esta dentro se ejecuta
```

```
while (numero <= 10) {
    document.write(numero + "<br>");
    numero++;
}
```

//Segundo caso

//Creamos nuestro array

```
var estudiantes = ["María", "Sergio", "Rosa", "Daniel"];
```

//Creamos la función que se ejecutará

```
function saludarEstudiantes(estudiante) {
    console.log(`Hola ${estudiante}`);
}
```

//Mientras la condición sea verdad la sentencia se va a ejecutar

```
while (estudiantes.length > 0) {
    console.log(estudiantes) //Vemos los elementos que quedan en el array
    var estudiante = estudiantes.shift(); //Se elimina el primer elemento
    saludarEstudiantes(estudiante); //Hace la funcion
}
```

//Ya al array llegar a 0 se rompe el bucle y lee la siguiente línea del código



Do While - Sintaxis

```
do
    sentencia
while (condición);

var i = 1;
do {
    document.write(i + "<br>");
    i++; //Ejecuta la sentencia al menos una
    vez hasta que valide la condición
} while (i <= 5); //Valida la condición y
si es verdadera la

//ejecuta hasta que deje de ser verdad
```

While y Do While

Do While

La sentencia (hacer mientras) crea un bucle que ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa. La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez.



Objects

Generación de objetos

JavaScript está diseñado en un paradigma simple basado en objetos. Un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor.

Generar un objeto

```
var miPerro = {  
  nombre : "Bruno",  
  raza : "Dalmata",  
  edad : 4,  
  genero : "Macho",  
  //propiedad = valor  
}
```

Acceder al objeto

```
miPerro.nombre  
// "Bruno"
```




Objects

Generación de objetos

*Añadir una
propiedad con
una función*

```
var miPerro = {  
  nombre: "Bruno",  
  raza: "Dalmata",  
  edad: "4",  
  genero: "Macho",  
  detallesDelPerro: function () {  
    console.log(`Perro ${this.raza} ${this.genero}`);  
    //El this significa que esta llamando al objeto (miPerro.raza) por ejemplo  
  },  
};  
  
miPerro.detallesDelPerro();  
//Perro Dalmata Macho
```

this = miPerro.



Objects

Función Constructora

Cuando creamos un objeto y guardamos sus propiedades, hay ciertas maneras de poder automatizar este proceso si queremos crear una "guía" para poder crear nuestros nuevos objetos sin hacerlo manualmente.

Primero **creamos la función** del objeto que servirá como plantilla

```
function estudiante(nombre,
apellido, edad) {
  this.nombre = nombre;
  this.apellido = apellido;
  this.edad = edad;
}
```

Creamos una variable que se guiará por la plantilla que nos permite elaborar nuevos objetos y desarrollar una nueva instancia.

```
var nuevoEstudiante = new estudiante("Luis",
  "Rosa", 21);
var nuevoEstudiante2 = new
  estudiante("Sebastian", "Arocha", 18);
//el operador new genera una nueva instancia
de la función constructora en un objeto
//instancia: objeto que deriva de otro objeto
```