

Introducción a Javascript

Programa “Introducción a la Programación Web”

Introducción a Javascript

- ☐ Usos
- ☐ Alcance de las variables
- ☐ Estructuras de control
- ☐ Funciones

Javascript - Introducción

JavaScript es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas.

Es el lenguaje de programación encargado de dotar de mayor interactividad y dinamismo a las páginas web.

A pesar de la similitud en el nombre, no está relacionado con Java.

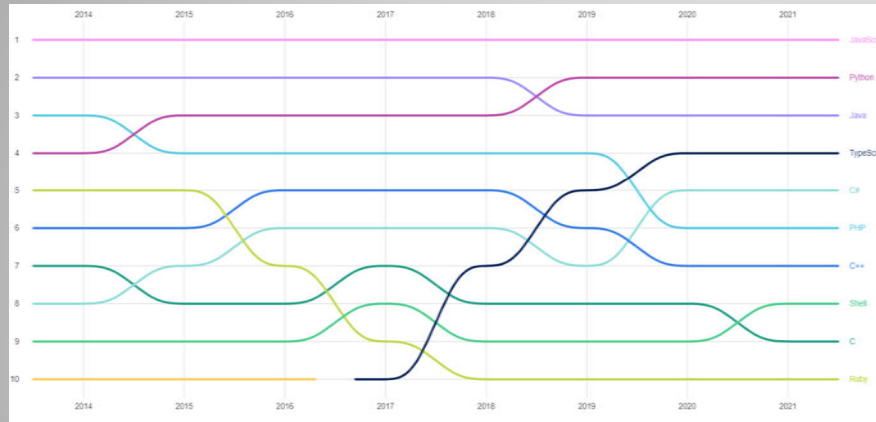
Javascript - Usos

A grandes rasgos, JavaScript te permite implementar **funciones** y crear **ambientes interactivos** en tus páginas web. Por ejemplo, permite mostrar y actualizar contenido, crear mapas interactivos, gráficos animados, realizar requests al servidor, etc.

En principio, corre del lado del cliente y es parte de las tres **tecnologías básicas** utilizadas para crear páginas web. (HTML, CSS, JS).

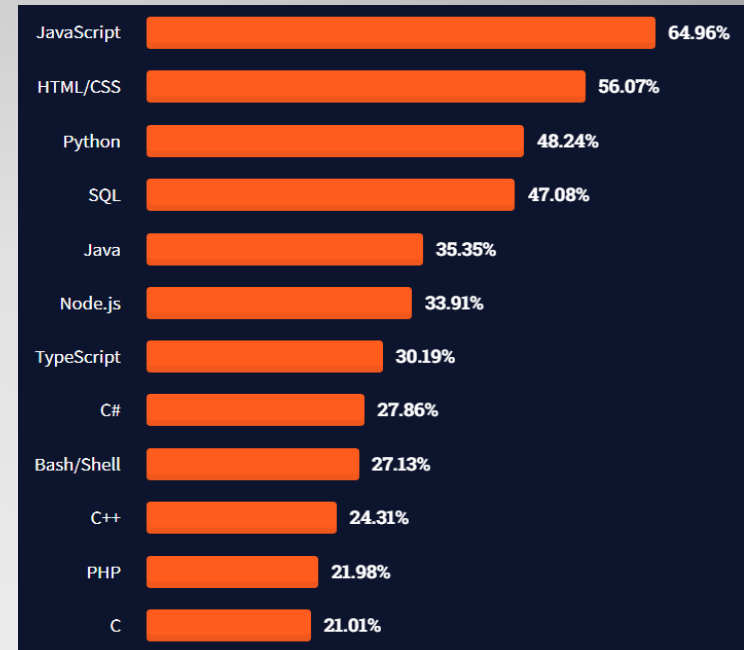


Javascript - ¿Cuánto se usa?



Fuente:

<https://octoverse.github.com>



Fuente:

<https://insights.stackoverflow.com/survey/2021>

Javascript - Algunas de sus funciones

Javascript del lado del cliente te permite realizar cosas como:

- Almacenar valores útiles en variables que pueden ser utilizados en la página.
- Operaciones con textos (strings), imágenes, y diferentes tipos de datos.
- Correr código en respuesta a ciertos eventos ocurriendo en la página web. Por ejemplo cuando realizamos un click, scrolleamos, cuando se carga la página, etc.
- Comunicarse con APIs, enviando y recibiendo información.

Javascript - Sintaxis

JavaScript está influenciado sobre todo por la sintaxis de Java, C y C++.

Distingue entre mayúsculas y minúsculas (es case-sensitive) y utiliza el conjunto de caracteres Unicode.

Todas las sentencias deben terminar con ;

¿Cómo debugear?

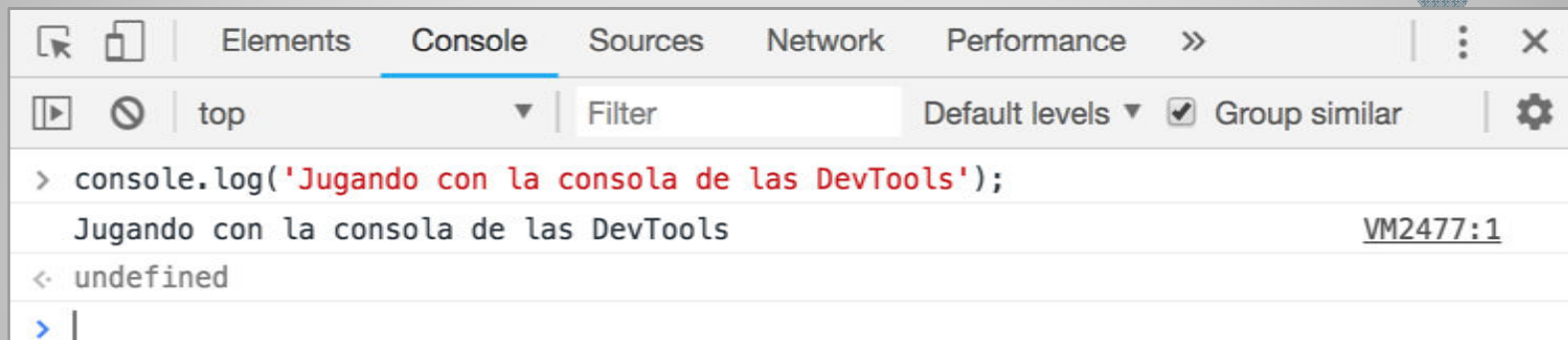


¿Dónde ver los datos!



“

La mejor manera es usando las DevTools y el **console.log()**





console.log



```
console.log("Hola mundo!");
```

```
// Luego se verá en las dev tools “Hola  
Mundo!”
```



¿Cómo vinculamos **JS** con
HTML?

Vinculación Interna

```
<body>
```

```
...
```

```
  <script>
```

```
    console.log("Hola mundo!");
```

```
  </script>
```

```
</body>
```

Vinculación Externa

```
<body>
```

```
...
```

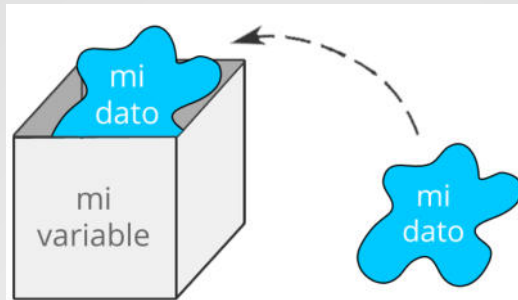
```
<script src="js/main.js"></script>
```

```
</body>
```

Javascript - Variables

Una variable es una unidad de **datos** que puede cambiar de valor.

Es la forma más simple de **almacenamiento**, representando una **zona de memoria** donde se almacena un elemento de datos.



Javascript - Tipos de Datos

Los tipos de datos que nos permiten usar son:

- Booleano (true, false).
- String ("Hola!").
- Number (256, 3.33, 0, -125).
- null (Denota que tiene valor nulo).
- undefined (Tipo de dato indefinido).
- Arrays
- Objetos

Javascript - Variables

Supongamos que queremos declarar una variable para almacenar diferentes nombres a lo largo de la ejecución. Podríamos hacerlo de la siguiente manera:

```
var unNombre;
```

La anterior sentencia se la conoce como “***declaración de variable***”.

Javascript - Variables

Ahora supongamos que queremos asignarle el nombre “Mónica” a la variable “unNombre”:

```
unNombre = “Mónica”;
```

A la anterior sentencia se la conoce como “***asignación de variable***”.

Javascript - Variables

Existen tres sentencias básicas que debemos conocer:

- **var:** declara variables con alcance local o global. Puede ser inicializada con algún valor.
- **let:** declara variables de alcance local, con ámbito de bloque. Puede ser inicializada con algún valor.
- **const:** los valores asignados a las constantes no pueden cambiarse, ni tampoco se pueden re-declarar. Tienen ámbito de bloque.

Javascript - Var vs. Let

Let te permite declarar variables limitando su alcance (scope) al bloque, declaración o expresión donde se esté usando.

Var define una variable global o local en una función sin importar el ámbito del bloque.

Javascript - Var vs. Let

```
var unNumero = 5;
var otroNumero = 10;
if (unNumero === 5) {
  let unNumero = 4; // El alcance es dentro del bloque if
  var otroNumero = 1; // El alcance es global
  console.log(unNumero ); // 4
  console.log(otroNumero ); // 1
}
console.log(unNumero ); // 5
console.log(otroNumero ); // 1
```

Javascript - Variables y tipos de datos

Las variables en JavaScript no están asociadas directamente con ningún tipo de valor en particular, y a cualquier variable se le puede asignar (y reasignar) valores de todos los tipos.

Ejemplo:

```
var miVariable = 42;    // miVariable ahora es un número  
  
miVariable      = 'bar'; // miVariable ahora es un string  
  
miVariable      = true;  // miVariable ahora es un booleano
```

Operadores Lógicos

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
	Operador or (o)	a b
!	Operador not (no)	!a

Operadores

× = // asignación

× + // suma

× - // resta

× * // multiplicación

× / // división

× % // módulo

× ++ // incremento

× -- // decremento

× == // igual simple

× === // igual

estricto

× != // distinto

× !== // distinto

estricto

× > // mayor

× < // menor

× >= // mayor o igual

× && // and

× || // or

× ! // negación

Estructuras de Control

¿Qué son? ¿Para que se utilizan?

Estructuras de Control - Tipos

- Secuenciales
- Selectivas
- Iterativas

Estructuras de Control - Secuencial

INSTRUCCIÓN 1

INSTRUCCIÓN 2

•

•

•

INSTRUCCIÓN N

Estructuras de Control - Selectivas

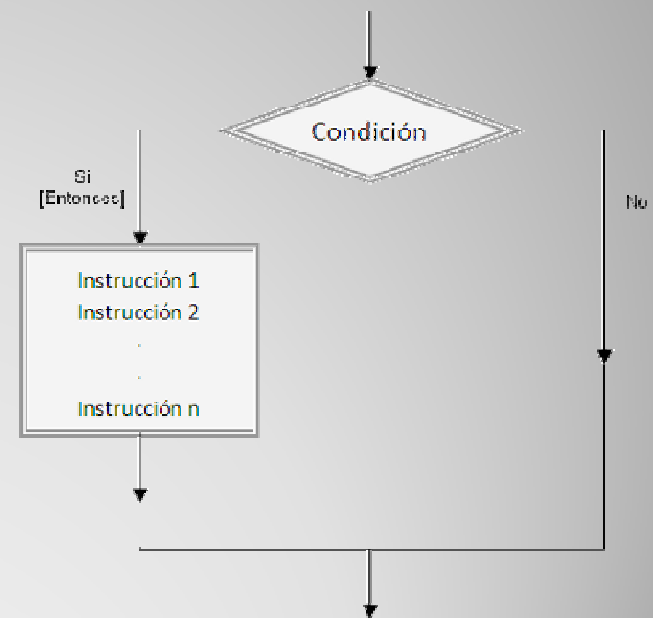
- **Simple**
- **Doble**
- **Múltiple o Anidada**

Estructuras de Control - Selectiva

Decisión Simple

Es la estructura de control más común. En esta, se obliga a evaluar una condición, que corresponde a expresiones lógicas.

Si la condición es **verdadera**, se ejecuta un conjunto de instrucciones. Si la condición es **falsa**, se ignoran y se continúa el programa después de la estructura.





if

```
if (true) {  
    // código a ejecutar when is true  
}
```

Estructuras de Control - Selectiva

Decisión Doble

Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición.

Se utiliza la declaración **“else”**, que indicará el código que debe ejecutarse en caso de que la condición no se cumpla. La declaración **“else”** no es obligatoria.





if / else

```
if (true) {  
    // código a ejecutar when is true  
} else {  
    // código a ejecutar when is false  
}
```

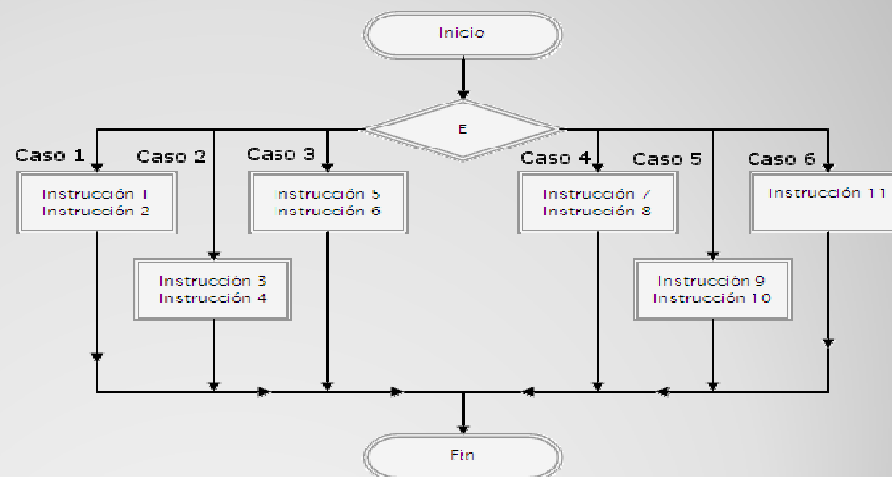
Estructuras de Control - Selectiva

Decisión Múltiple o anidada

Las estructuras condicionales múltiples permiten combinar selectivas simples y dobles para crear estructuras más complejas.

Estructuras de Control - Switch

La estructura selectiva switch selecciona una de entre múltiples alternativas. Esta estructura es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada expresión de control o selector.



Estructuras de Control - Switch

```
switch (expresión) {  
    case valor1:  
        //Declaraciones ejecutadas cuando el resultado de expresión coincide con el valor1  
        [break;]  
    case valor2:  
        //Declaraciones ejecutadas cuando el resultado de expresión coincide con el valor2  
        [break;]  
    ...  
    case valorN:  
        //Declaraciones ejecutadas cuando el resultado de expresión coincide con valorN  
        [break;]  
    default:  
        //Declaraciones ejecutadas cuando ninguno de los valores coincide con el valor de la expresión  
        [break;]  
}
```

Estructuras de Control - Iterativa

Con cantidad fija de iteraciones

Se utilizan cuando a priori se conoce la cantidad de ocasiones que debe repetirse un bloque de instrucciones.

PARA <VARIABLE> DESDE <VALOR1> HASTA <VALOR2> CON PASO
<VALOR3> HACER

ACCIÓN/ES

FIN PARA

Estructuras de Control - Iterativa

Con cantidad variable de iteraciones

Se utilizan cuando la cantidad de ocasiones que debe repetirse un bloque de instrucciones está determinado por una condición.

MIENTRAS QUE <CONDICIÓN> HACER
ACCIÓN/ES A REPETIR
FIN MIENTRAS

REPETIR
ACCIÓN/ES
HASTA QUE <CONDICIÓN>

Bucles - for



```
for (var i = 0; i < 10; i++) {  
  // código a ejecutar.  
}
```

Bucles - while

```
while ( CONDICION ) {  
    // código a ejecutar.  
}
```



Bucles – do while

```
do {  
    // código a ejecutar.  
} while ( CONDICION )
```

La diferencia entre con el while es que al menos se ejecuta la primera acción (primer ciclo) independientemente si se cumple la condición del while

Bucles – for

```
for (var elemento of array) {  
    console.log(elemento);  
}
```

Este tipo de bucle permite recorrer un array, donde **elemento** es una variable nueva que representa cada elemento del **array** que debe ser un array

Bucles - for

```
for (var elemento in objeto) {  
    console.log(elemento);  
}
```

Este tipo de bucle permite recorrer un array, donde **elemento** es una variable nueva que representa cada atributo de un **objeto** que debe ser un objeto

OBJETOS LITERALES

Representación en código de un
elemento de la vida real

Objetos literales

¿Qué es?

Un objeto de JavaScript es un bloque de código que tiene **propiedades**, las cuales a su vez tienen un valor determinado.

Si una propiedad recibe como valor una **función**, a esto lo llamamos un **método**.

Objetos literales

```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
}
```

Ahora ¿cómo accedemos al nombre del estudiante?



“

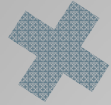
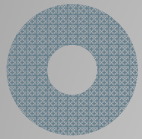
Con la notación **objeto.prop**,
accedemos al valor de dicha
propiedad.

A black horizontal bar at the top of the slide contains several white geometric shapes: a plus sign, a cross, a circle, a square, a diamond, and another circle.

Objetos literales

```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
}
```

```
console.log(student.name); // "Juana"
```




¿Cómo se ven los **métodos**
de un objeto?

Objetos literales

```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
  fullName: function () {  
    return student.name + " " + student.lastName;  
  }  
}
```

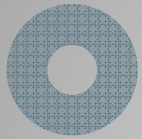



Y ahora:
¿Cómo podemos ejecutar
dicho método?



Objetos literales

```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
  fullName: function () {  
    return student.name + " " + student.lastName;  
  }  
}  
console.log(student.fullName()); // "Juana Heinz"
```



“

¿Qué sucede con la notación
object.prop dentro de un
método del objeto si el
nombre del objeto cambia?



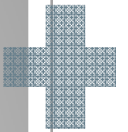
Objetos literales

```
var juanitaEstudiante = {  
  name: "Juana",  
  lastName: "Heinz",  
  fullName: function () {  
    return this.name + " " + this.lastName;  
  }  
}  
console.log(juanitaEstudiante.fullName()); // "Juana Heinz"
```

“

La palabra reservada **this** hace referencia al mismo objeto en sí.

Se usa bastante incluso en otros ámbitos que no sean objetos literales.





“

Y, si un objeto literal tiene
muchas propiedades
¿qué podemos hacer para
acceder a todas ellas?



Objetos literales

```
for (var key in student) {  
    console.log(student[key]);  
}
```

```
// "Juana"  
// "Heinz"  
// "f()" ¿?
```



“

El **for in** es un bucle que
nos permite **iterar** sobre cada
una de las propiedades de un
objeto literal.

Funciones

¿Qué son las funciones?

Funciones

- Una función es un conjunto de líneas de código que realizan una **tarea específica** y puede **retornar un valor**.
- Las funciones pueden tomar **parámetros** que modifiquen su funcionamiento.
- Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera **reducir la cantidad de código**.

¿Qué diferencia hay entre “función” y “procedimiento”?

Diferencia entre una función y un procedimiento

Una **función** es una sección de un programa que calcula un valor de manera independiente al resto del programa.

En resumen, una función es un mini programa: tiene una entrada, un proceso y una salida.

Una función tiene tres componentes importantes:

- Los **parámetros**, que son los valores que recibe la función como entrada;
- El **código de la función**, que son las operaciones que realiza la función; y
- El **resultado** o **valor de retorno**, que es el valor final que entrega la función.

Un **procedimiento** es una sección de un programa (al igual que una función) que realiza varias sentencias de manera independiente al resto del programa.

La diferencia con una función es que un procedimiento no entrega ningún valor como resultado. Los procedimientos son útiles para agrupar secuencias de sentencias que deben ser realizadas juntas.

Usar procedimientos suele hacer que los programas sean más fáciles de leer.

¿Cómo declaramos una función?

Una función es un bloque de código definido para realizar una acción en específica.

```
function nombre(parametro1, parametro2, parametro3) {  
    // código a ejecutar.  
}
```

- Los **parámetros** de una función son listados dentro de los paréntesis ()
- Los **argumentos** de una función son los **valores** recibidos cuando la función es invocada.
- Dentro de la función, los **argumentos** (parámetros) actúan como variables locales.

Referencias

- [Variables - Javascript](#)
- [Funciones - Javascript](#)
- [Ciclos de Repetición - Javascript](#)
- [Estructuras de Control - Javascript](#)

¿Preguntas?

Gracias!