



CJAVA

The logo consists of a stylized 'C' and 'JAVA' wordmark. Above the 'C', there is a graphic element composed of three interlocking 3D cubes forming a cube-like shape.



**CJAVA**  
siempre para apoyarte

010101010101010101010101010101  
001010101010101010101010101010101  
0101010101010101010101010101010101  
01010101010101010101  
1010101010101010  
0101010101010101010101010101010101  
101010101010101010101001010010101010101  
01010101010101

## Visión

Poder aportar al desarrollo del País usando tecnología Java.

# Quienes Somos

Somos una organización orientada a **desarrollar, capacitar e investigar tecnología JAVA** a través de un prestigioso staff de profesionales a nivel nacional.





cjavaperu.com  
info@cjavaperu.com

COMUNIDAD

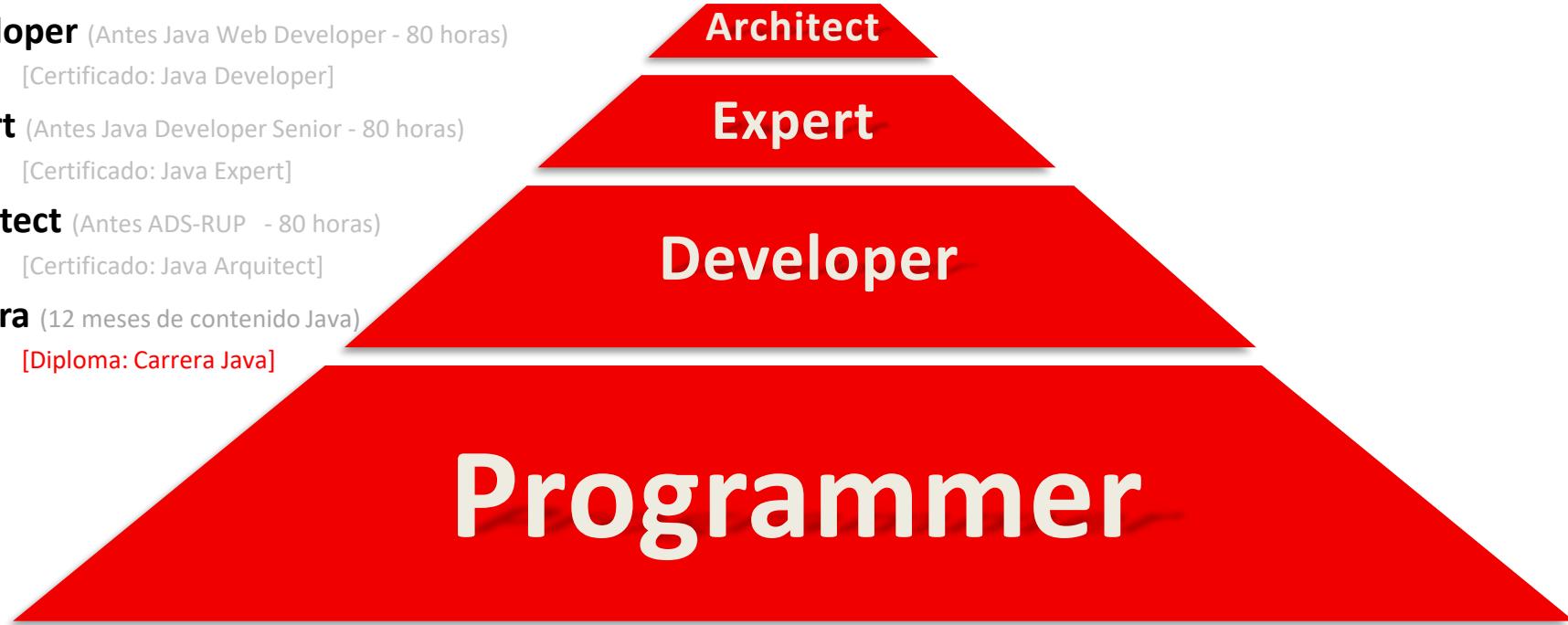
ACADEMICA

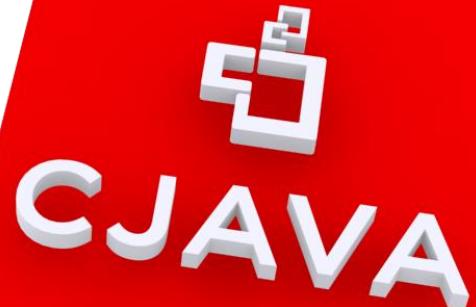


siempre para apoyarte

# Servicio de Capacitación

- **Programer** (Antes Java Developer Junior - 80 horas)  
[Certificado: Java Programer]
- **Developer** (Antes Java Web Developer - 80 horas)  
[Certificado: Java Developer]
- **Expert** (Antes Java Developer Senior - 80 horas)  
[Certificado: Java Expert]
- **Arquitect** (Antes ADS-RUP - 80 horas)  
[Certificado: Java Arquitect]
- **Carrera** (12 meses de contenido Java)  
[Diploma: Carrera Java]

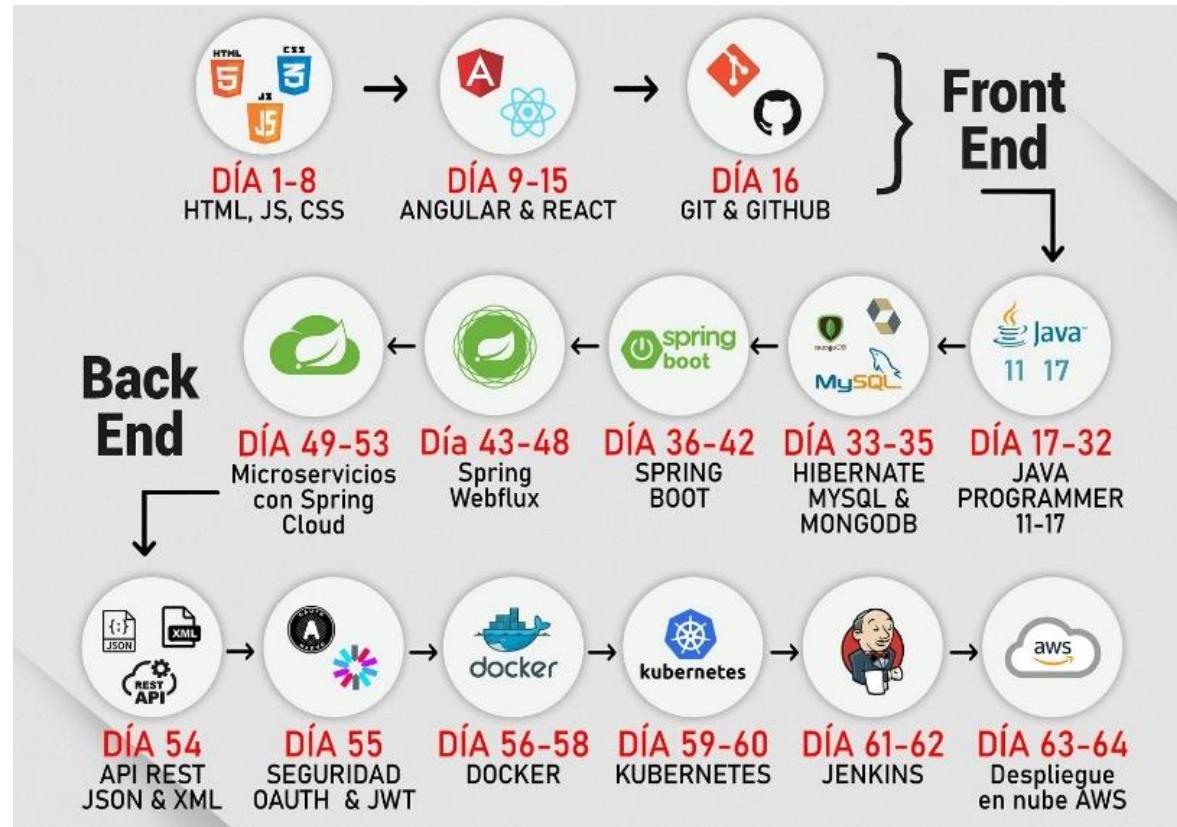




# Java FullStack Developer

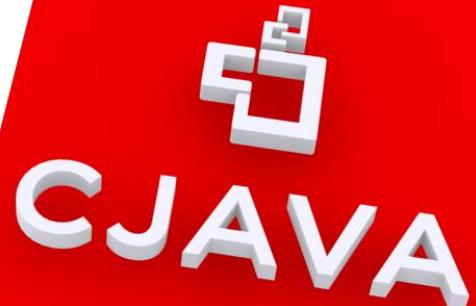
emaravi@cjavaperu.com

# Introducción a Full Stack Development



# Introducción a Full Stack Development

Front-end Development: Discutir las tecnologías y herramientas utilizadas en el desarrollo front-end, como HTML, CSS, JavaScript y React.



# JavaScript

Edwin Maravi

# Resultado de aprendizaje

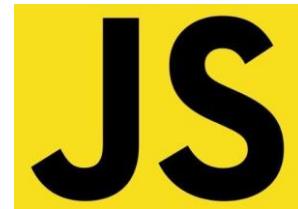
Al finalizar la sesión, el estudiante elabora aplicaciones web con JavaScript reconociendo los tipos de datos primitivos, variables y el uso de operadores.

# Contenidos o temas

- Introducción a JavaScript
- Tipos de datos y estructuras
- Declaración de variables
- Operadores
- Funciones predefinidas

# Introducción a JavaScript

- JavaScript (abreviado comúnmente **JS**) es un lenguaje de programación interpretado, dialecto del estándar **ECMAScript**. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
- Se utiliza principalmente del **lado del cliente**, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas y JavaScript del **lado del servidor** (Server-side JavaScript o SSJS).

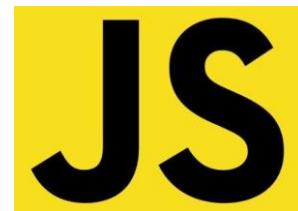


# Introducción a JavaScript

- JavaScript se diseñó con una sintaxis similar a **C++ y Java**, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.
- Todos los **navegadores modernos** interpretan el código JavaScript integrado en las páginas web.
- **Tradicionalmente** se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor.

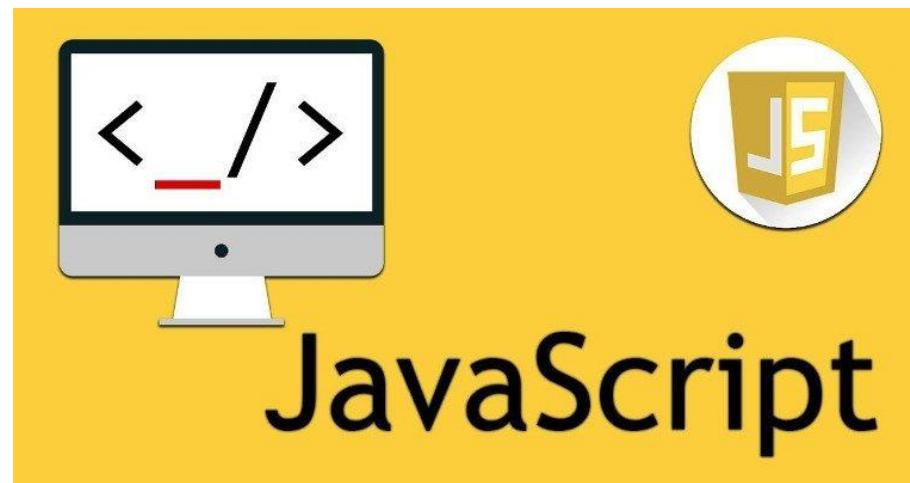
# Introducción a JavaScript

- Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como **AJAX**.
- A partir de mediados de la década de los 2000, ha habido una proliferación de implementaciones de JavaScript para el lado servidor **Node.js** es uno de los notables ejemplos de JavaScript en el lado del servidor, siendo usado en proyectos importantes.



# Introducción a JavaScript

Es la tercera capa del pastel de las tecnologías web estándar, dos de las cuales (**HTML y CSS**) hemos cubierto con mucho más detalle en otras partes del Área de aprendizaje.



# Introducción a JavaScript

- **HTML** es el lenguaje de marcado que usamos para estructurar y dar significado a nuestro contenido web, por ejemplo, definiendo párrafos, encabezados y tablas de datos, o insertando imágenes y videos en la página.
- **CSS** es un lenguaje de reglas de estilo que usamos para aplicar estilo a nuestro contenido HTML, por ejemplo, establecer colores de fondo y tipos de letra, y distribuir nuestro contenido en múltiples columnas.
- **JavaScript** es un lenguaje de secuencias de comandos que te permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes y prácticamente todo lo demás.



# Introducción a JavaScript

¿Para qué se utiliza JavaScript?

Actualización dinámica de contenidos.

Desarrollo de aplicaciones web y móviles.

Desarrollo de aplicaciones del lado del servidor.

Desarrollo de videojuegos



# Tipos de datos y estructuras en JavaScript

- Todos los lenguajes de programación tienen estructuras de datos integradas, pero estas a menudo difieren de un lenguaje a otro.
- 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:

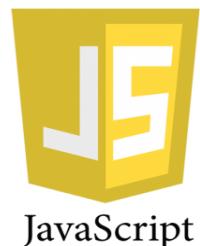
```
let foo = 42;      // foo ahora es un número
foo      = 'bar'; // foo ahora es un string
foo      = true;  // foo ahora es un booleano
```

# Tipos de datos y estructuras en JavaScript

- El último estándar **ECMAScript** define nueve tipos.
- Seis **tipos de datos primitivos**, controlados por el operador **typeof**:
  - **Undefined**: `typeof instance === "undefined"`. Una variable a la que no se le ha asignado un valor tiene el valor undefined.
  - **Boolean**: `typeof instance === "boolean"`. Representa una entidad lógica y puede tener dos valores: true y false.
  - **Number**: `typeof instance === "number"`. Es un valor en formato binario de 64 bits de doble precisión IEEE 754.
  - **String**: `typeof instance === "string"`. Se utiliza para representar datos textuales. Es un conjunto de "elementos" de valores enteros sin signo de 16 bits.
  - **BigInt**: `typeof instance === "bigint"`. Almacena y opera de forma segura números enteros grandes incluso más allá del límite seguro de enteros para Numbers.
  - **Symbol**: `typeof instance === "symbol"`. Un símbolo es un valor primitivo único e inmutable y se puede utilizar como clave de una propiedad de objeto.

# Tipos de datos y estructuras en JavaScript

- **Null: typeof instance === "object"**. Tipo primitivo **especial** que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra null;
- **Object: typeof instance === "object"**. Tipo **estructural** especial que no es de datos pero para cualquier instancia de objeto construido que también se utiliza como estructuras de datos: new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date y casi todo lo hecho con la palabra clave new.
- **Function: typeof instance === "function"**. Una **estructura** sin datos, aunque también responde al operador typeof. Esta simplemente es una forma abreviada para funciones, aunque cada constructor de funciones se deriva del constructor Object.



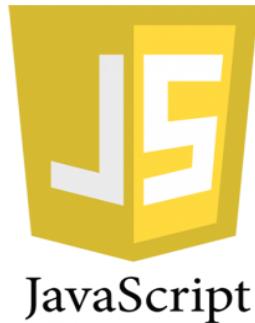
# Declaración de variables en JavaScript

JavaScript tiene **tres tipos** de declaraciones de variables.

**var:** Declara una variable, opcionalmente la inicia a un valor.

**let:** Declara una variable local con ámbito de bloque, opcionalmente la inicia a un valor.

**const:** Declara un nombre de constante de solo lectura y ámbito de bloque.



# Declaración de variables en JavaScript

- Utiliza **variables** como nombres simbólicos para valores en tu aplicación. Los nombres de las variables, llamados **identificadores**, se ajustan a ciertas reglas.
- Un **identificador** de JavaScript debe comenzar con una letra, un guión bajo (\_) o un signo de dólar (\$). Los siguientes caracteres también pueden ser dígitos (0-9).
- Dado que JavaScript distingue entre **mayúsculas y minúsculas**, las letras incluyen los caracteres "A" a "Z" (mayúsculas), así como "a" a "z" (minúsculas).



# Declaración de variables en JavaScript

- Puedes declarar una variable de **dos formas**:
- Con la **palabra clave var**. Por ejemplo, `var x = 42`. Esta sintaxis se puede utilizar para declarar variables **locales y globales**, dependiendo del contexto de ejecución.
- Con la **palabra clave const o let**. Por ejemplo, `let y = 13`. Esta sintaxis se puede utilizar para declarar una **variable local** con ámbito de bloque.



# Operadores en JavaScript

- JavaScript tiene los siguientes tipos de operadores. Esta sección describe los operadores y contiene información sobre la precedencia de los mismos.
- **Aritméticos**
- **Lógicos**
- **Asignación**
- **Comparación**

{.js}  
**JavaScript**

# Operadores en JavaScript

- Operadores Aritméticos

**{.js}**  
**JavaScript**

| Operador            | Descripción  | Ejemplo  |
|---------------------|--|--|
| Resto (%)           | Operador binario correspondiente al módulo de una operación. Devuelve el resto de la división de dos operandos.  | <code>12 % 5</code> devuelve 2.  |
| Incremento (++)     | Operador unario. Incrementa en una unidad al operando. Si es usado antes del operando (++x) devuelve el valor del operando después de añadirle 1 y si se usa después del operando (x++) devuelve el valor de este antes de añadirle 1. | Si x es 3, entonces ++x establece x a 4 y devuelve 4, mientras que x++ devuelve 3 y, solo después de devolver el valor, establece x a 4. |
| Decremento (--)     | Operador unario. Resta una unidad al operando. Dependiendo de la posición con respecto al operando tiene el mismo comportamiento que el operador de incremento.  | Si x es 3, entonces --x establece x a 2 y devuelve 2, mientras que x-- devuelve 3 y, solo después de devolver el valor, establece x a 2. |
| Negación Unaria (-) | Operación unaria. Intenta convertir a número al operando y devuelve su forma negativa.   | <code>-"3"</code> devuelve -3.<br><code>-true</code> devuelve -1.  |
| Unario positivo (+) | Operación unaria. Intenta convertir a número al operando.  | <code>+"3"</code> devuelve 3.<br><code>+true</code> devuelve 1.  |
| Exponenciación (**) | Calcula la potencia de la base al valor del exponente. Es equivalente a <code>base<sup>exponente</sup></code>  | <code>2 ** 3</code> devuelve 8.<br><code>10 ** -1</code> devuelve 0.1.   |

# Operadores en JavaScript

- Operadores Lógicos

| Operador                                  | Uso                                 | Descripción  |
|---|-------------------------------------|--|
| AND Lógico<br>( <code>&amp;&amp;</code> ) | <code>expr1 &amp;&amp; expr2</code> | Devuelve <code>expr1</code> si puede ser convertido a <code>false</code> de lo contrario devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>&amp;&amp;</code> devuelve <code>true</code> si ambos operandos son <code>true</code> , en caso contrario devuelve <code>false</code> .      |
| OR Lógico<br>( <code>  </code> )          | <code>expr1    expr2</code>         | Devuelve <code>expr1</code> si puede ser convertido a <code>true</code> de lo contrario devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>  </code> devuelve <code>true</code> si alguno de los operandos es <code>true</code> , o <code>false</code> si ambos son <code>false</code> . |
| NOT<br>Lógico<br>( <code>!</code> )       | <code>!expr</code>                  | Devuelve <code>false</code> si su operando puede ser convertido a <code>true</code> , en caso contrario, devuelve <code>true</code> .  |

{.js}  
JavaScript

# Operadores en JavaScript

- Operadores Asignación

| Nombre                       | Operador abreviado   | Significado             |
|------------------------------|----------------------|-------------------------|
| Operadores de asignación     | <code>x = y</code>   | <code>x = y</code>      |
| Asignación de adición        | <code>x += y</code>  | <code>x = x + y</code>  |
| Asignación de sustracción    | <code>x -= y</code>  | <code>x = x - y</code>  |
| Asignación de multiplicación | <code>x *= y</code>  | <code>x = x * y</code>  |
| Asignación de división       | <code>x /= y</code>  | <code>x = x / y</code>  |
| Asignación de resto          | <code>x %= y</code>  | <code>x = x % y</code>  |
| Asignación de exponenciación | <code>x **= y</code> | <code>x = x ** y</code> |

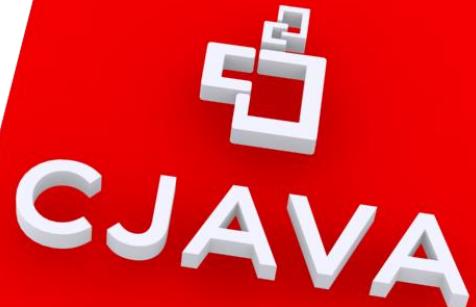
# Operadores en JavaScript

- Operadores de Comparación

| Operador                         | Descripción  | Ejemplos devolviendo true   |
|----------------------------------|--|---|
| Igualdad ( == )                  | Devuelve <code>true</code> si ambos operandos son iguales.   | <code>3 == var1</code><br><code>"3" == var1</code><br><code>3 == "3"</code> |
| Desigualdad ( != )               | Devuelve <code>true</code> si ambos operandos no son iguales.  | <code>var1 != 4</code><br><code>var2 != "3"</code>                          |
| Estrictamente iguales ( === )    | Devuelve <code>true</code> si los operandos son igual y tienen el mismo tipo.<br>Mira también <a href="#">Object.is</a> y <a href="#">sameness in JS</a> . | <code>3 === var1</code>   |
| Estrictamente desiguales ( !== ) | Devuelve <code>true</code> si los operandos no son iguales y/o no son del mismo tipo.  | <code>var1 !== "3"</code><br><code>3 !== "3"</code>                         |
| Mayor que ( > )                  | Devuelve <code>true</code> si el operando de la izquierda es mayor que el operando de la derecha.  | <code>var2 &gt; var1</code><br><code>"12" &gt; 2</code>                     |
| Mayor o igual que ( >= )         | Devuelve <code>true</code> si el operando de la izquierda es mayor o igual que el operando de la derecha.  | <code>var2 &gt;= var1</code><br><code>var1 &gt;= 3</code>                   |
| Menor que ( < )                  | Devuelve <code>true</code> si el operando de la izquierda es menor que el operando de la derecha.  | <code>var1 &lt; var2</code><br><code>"2" &lt; 12</code>                     |
| Menor o igual que ( <= )         | Devuelve <code>true</code> si el operando de la izquierda es menor o igual que el operando de la derecha.  | <code>var1 &lt;= var2</code><br><code>var2 &lt;= 5</code>                   |

# Funciones predefinidas en JavaScript

- **eval(string):** Esta función recibe una cadena de caracteres y la ejecuta como si fuera una sentencia de Javascript.
- **parseInt(cadena, base):** Recibe una cadena y una base. Devuelve un valor numérico resultante de convertir la cadena en un número en la base indicada.
- **parseFloat(cadena):** Convierte la cadena en un número y lo devuelve.
- **isNaN (número):** Devuelve un booleano dependiendo de lo que recibe por parámetro. Si no es un número devuelve un true, si es un número devuelve false.
- **escape(string):** Crea una nueva cadena de caracteres en los que ciertos caracteres han sido sustituidos por una secuencia hexadecimal de escape.



# JavaScript

EDWIN MARAVI

CJava, siempre para apoyarte.

# Contenidos o temas

- Estructuras de control de flujo:  
condicional y cíclico.
- Anidamiento entre estructuras de  
control de flujo.

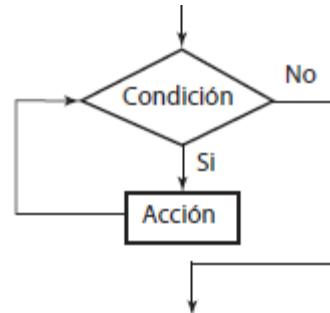
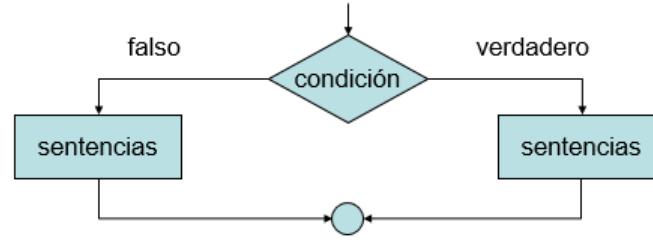
# Estructuras de control con JavaScript

Las estructuras de control de flujo, **son instrucciones** que nos permiten **evaluar** si se puede **cumplir** una condición o no, incluso nos puede ayudar a evaluarla “n” cantidad de veces.



# Estructuras de control con JavaScript

Para realizar este tipo de programas son necesarias las estructuras de control de flujo, que son instrucciones del tipo "**si** se cumple esta condición, hazlo; si no se cumple, haz esto otro". También existen instrucciones del tipo "**repite** esto mientras se cumpla esta condición".



# Estructura de control if en JavaScript

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la **estructura if**. Se emplea para **tomar decisiones** en función de una **condición**. Su **definición formal** es:

```
if(condicion) {  
    ...  
}
```

# Estructura de control if en JavaScript

- Si la **condición se cumple** (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.
- **Ejemplo:**

```
var mostrarMensaje = true;

if(mostrarMensaje == true) {
    alert("Hola Mundo");
}
```

# Estructura de control if..else en JavaScript

- En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". **Normalmente** las condiciones suelen ser del tipo "*si se cumple esta condición, hazlo; si no se cumple, haz esto otro*".
- Para este segundo tipo de decisiones, existe una variante de la estructura if llamada **if...else**. Su **definición formal** es la siguiente:

```
if(condicion) {  
    ...  
}  
else {  
    ...  
}
```

# Estructura de control if..else en JavaScript

- Si la **condición se cumple** (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else {}.
- **Ejemplo:**

```
var edad = 18;

if(edad >= 18) {
    alert("Eres mayor de edad");
}

else {
    alert("Todavía eres menor de edad");
}
```

# Estructura de control if (Anidamiento)

- La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {  
    alert("Todavía eres muy pequeño");  
}  
else if(edad < 19) {  
    alert("Eres un adolescente");  
}  
else if(edad < 35) {  
    alert("Aun sigues siendo joven");  
}  
else {  
    alert("Piensa en cuidarte un poco más");  
}
```



# Estructura de control switch en JavaScript

Una instrucción **switch** permite que un programa evalúe una **expresión** e intente hacer coincidir el valor de la expresión con una **etiqueta case**. Si la encuentra, el programa ejecuta la declaración asociada. La **definición** del switch se ve así:

```
switch (expression) {  
    case label_1:  
        statements_1  
        [break;]  
    case label_2:  
        statements_2  
        [break;]  
        ...  
    default:  
        statements_def  
        [break;]  
}
```



# Estructura de control switch en JavaScript

Es el caso típico de los **menús** de elección de opciones. En función de la **opción** elegida por el usuario nosotros debemos hacer lo que nos pide, como se muestra en el siguiente **ejemplo**:

```
switch(num)
{
    case 1:
        document.write("Has elegido el numero 1");
        break;
    case 2:
        document.write("Has elegido el numero 2");
        break;
    case 3:
        document.write("Has elegido el numero 3");
        break;
    default:
        document.write("Has elegido un numero diferente que no esta en el intervalo de 1 a 3");
        break;
}
```

# Estructura de control for en JavaScript

El **bucle FOR** se utiliza para **repetir** una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando **sabemos** seguro el número de veces que queremos que se ejecute. La **sintaxis** del bucle for se muestra a continuación.

```
for(inicializacion; condicion; actualizacion) {  
    ...  
}
```

# Estructura de control for en JavaScript

- La "**inicialización**" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "**condición**" es el único elemento que decide si continua o se detiene la repetición.
- La "**actualización**" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.
- **Ejemplo:**

```
var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {
    alert(mensaje);
}
```

# Estructura de control for...in en JavaScript

Una estructura de control derivada de for es la **estructura for...in**. Su definición exacta implica el **uso de objetos**, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura for...in adaptada a su uso en **arrays**. Su **definición formal** adaptada a los arrays es:

```
for(indice in array) {  
    ...  
}
```



# Estructura de control for...in en JavaScript

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente **ejemplo**:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];  
  
for(i in dias) {  
    alert(dias[i]);  
}
```

# Estructura de control while en JavaScript

El ciclo while recorre un bloque de código **mientras** que una condición específica **sea verdadera**, su sintaxis es la siguiente:

```
while (condition) {  
    // code block to be executed  
}
```

# Estructura de control while en JavaScript

En el siguiente **ejemplo**, el código del ciclo se ejecutará una y otra vez, mientras que la variable (i) sea menor que 10:

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```



# Estructura de control do...while en JavaScript

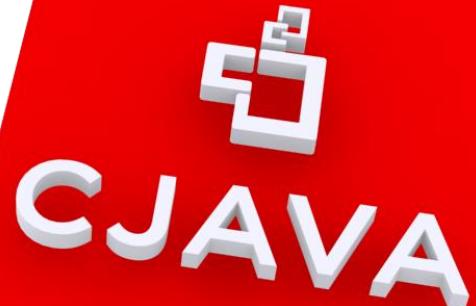
El bucle **do while** es una variante del bucle while. Este ciclo ejecutará el bloque de código una vez, antes de verificar si la condición es verdadera, luego repetirá el ciclo **mientras** la condición sea verdadera, su **sintaxis** es la siguiente:

```
do {  
    // code block to be executed  
}  
while (condition);
```

# Estructura de control do...while en JavaScript

El siguiente **ejemplo** utiliza un bucle do while. El bucle siempre se ejecutará al menos una vez, incluso si la condición es falsa, porque el bloque de código se ejecuta antes de que se pruebe la condición:

```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```



# JavaScript

EDWIN MARAVI

# Contenidos o temas

- Creación de funciones de usuario
- Funciones con y sin parámetros
- Llamada de funciones
- Valor de retorno de una función

# Funciones de usuario en JavaScript

- Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las **mismas instrucciones**.
- Las funciones son la **solución** a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. Una función es un conjunto de instrucciones que se **agrupan** para realizar una tarea concreta y que se pueden **reutilizar fácilmente**.



# Funciones de usuario en JavaScript

Las funciones son uno de los **bloques de construcción** fundamentales en JavaScript. Una función en JavaScript es similar a un **procedimiento** — un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función, debe tomar alguna **entrada** y devolver una **salida** donde hay alguna relación obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamarla.



# Funciones de usuario sin parámetros (argumentos) en JavaScript

En primer lugar se debe crear la **función básica** con las **instrucciones comunes**. Las funciones en JavaScript se definen mediante la palabra reservada **function**, seguida del **nombre** de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {  
    ...  
}
```

# Funciones de usuario sin parámetros (argumentos) en JavaScript

Se **crea** una función llamada `suma_y_muestra()` de la siguiente forma:

```
function suma_y_muestra() {  
    resultado = numero1 + numero2;  
    alert("El resultado es " + resultado);  
}
```

# Funciones de usuario sin parámetros (argumentos) en JavaScript

La **Llamada** a la función se realiza simplemente indicando su nombre, incluyendo los paréntesis del final y el carácter ; para terminar la instrucción:

```
function suma_y_muestra() {  
    resultado = numero1 + numero2;  
    alert("El resultado es " + resultado);  
}  
  
var resultado;  
  
var numero1 = 3;  
var numero2 = 5;  
  
suma_y_muestra();  
  
numero1 = 10;  
numero2 = 7;  
  
suma_y_muestra();
```

# Funciones de usuario con parámetros (argumentos) en JavaScript

La palabra clave **function** va primero, luego va el **nombre de función**, luego una **lista de parámetros** entre paréntesis (separados por comas, vacía en el ejemplo anterior) y finalmente el código de la función entre **llaves {}**, también llamado “**el cuerpo de la función**”.

```
function name(parameter1, parameter2, ... parameterN) {  
    // body  
}
```

# Funciones de usuario con parámetros (argumentos) en JavaScript

- Podemos pasar datos arbitrarios a funciones usando parámetros.
- En el siguiente **ejemplo**, la función tiene dos parámetros: **from** y **text**. Cuando la función se llama (\*) y (\*\*), los valores dados se **copian** en variables locales from y text. Y la función las utiliza.

```
function showMessage(from, text) { // parámetros: from, text
    alert(from + ': ' + text);
}

showMessage('Ann', '¡Hola!'); // Ann: ¡Hola! (*)
showMessage('Ann', "¿Cómo estás?"); // Ann: ¿Cómo estás? (**)
```

# Funciones de usuario devolviendo un valor (Retorno)

- Una función puede **devolver** un valor al código de llamada como **resultado**.
- El **ejemplo** más simple sería una función que suma dos valores:

```
function sum(a, b) {  
    return a + b;  
}  
  
let result = sum(1, 2);  
alert(result); // 3
```

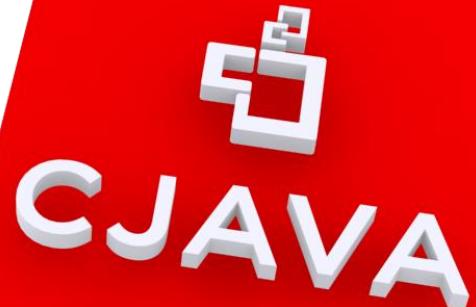
# Funciones de usuario devolviendo un valor (Retorno)

- La directiva **return** puede estar en cualquier lugar de la función. Cuando la ejecución lo alcanza, la función se detiene y el valor se devuelve al código de llamada (asignado al result anterior).
- Puede haber muchos casos de **return** en una sola función. Por ejemplo:

```
function checkAge(age) {
    if (age > 18) {
        return true;
    } else {
        return confirm('¿Tienes permiso de tus padres?');
    }
}

let age = prompt('¿Qué edad tienes?', 18);

if ( checkAge(age) ) {
    alert( 'Acceso otorgado' );
} else {
    alert( 'Acceso denegado' );
}
```



# JavaScript

EDWIN MARAVI

# Contenidos o temas

- Objeto String
- Objeto Date
- Objeto Math

# Objeto String

- El objeto String se utiliza para representar y manipular una secuencia de **caracteres**.
- Las **cadenas** son útiles para almacenar datos que se pueden representar en forma de texto. Algunas de las operaciones más utilizadas en cadenas son verificar su *length*, para construirlas y concatenarlas usando operadores de cadena + y +=, verificando la existencia o ubicación de subcadenas con *indexOf()* o extraer subcadenas con el método *substring()*.



# Objeto String

- Las cadenas se pueden crear como primitivas, a partir de cadena literales o como objetos, usando el constructor String():

```
const string1 = "Una cadena primitiva";
const string2 = 'También una cadena primitiva';
const string3 = `Otra cadena primitiva más`;
```

```
const string4 = new String("Un objeto String");
```



# Propiedades del Objeto String

La clase String sólo tiene una propiedad: **length**, que guarda el número de caracteres del String.

| PROPIEDAD | QUÉ HACE                                |
|-----------|---|
| length    | Corresponde a la longitud de la cadena. |



# Métodos del Objeto String

Los objetos de la clase String tienen una buena cantidad de **métodos**, para realizar lo siguiente:

| MÉTODO                                  | QUÉ HACE   |
|---|--|
| <code>charAt(num)</code>                | Permite acceder a un carácter en concreto de una cadena.   |
| <code>indexOf(string)</code>            | Devuelve la posición de la primera ocurrencia del carácter pasado como parámetro.                      |
| <code>lastIndexOf(string)</code>        | Devuelve la posición de la última ocurrencia del carácter pasado como parámetro                        |
| <code>match()</code>                    | Busca una coincidencia en una cadena y devuelve todas las coincidencias encontradas                    |
| <code>replace(cadena, sustituto)</code> | Busca una coincidencia en una cadena y si existe, la remplaza por otra cadena pasada como parámetro    |
| <code>search()</code>                   | Busca una coincidencia en una cadena y devuelve la posición de la coincidencia                         |
| <code>slice()</code>                    | Extrae una parte de una cadena en base a los parámetros que indiquemos como índices de inicio y final. |

# Métodos del Objeto String

|                                 |   |
|---------------------------------|---|
| <b>split()</b>                  | Corta una cadena en base a un separador que pasamos como parámetro                      |
| <b>substr(inicio, longitud)</b> | Devuelve una subcadena en base a un índice y longitud pasados como parámetros           |
| <b>substring(inicio, fin)</b>   | Devuelve una subcadena en base a un índice de inicio y de final pasados como parámetros |
| <b>toLowerCase()</b>            | Devuelve la cadena en minúsculas. No la cambia.   |
| <b>toUpperCase()</b>            | Devuelve la cadena en mayúsculas. No la cambia  |
| <b>trim()</b>                   | Elimina los espacios del principio y el final del String                                |
| <b>fromCharCode()</b>           | Convierte valores unicode en caracteres   |
| <b>concat()</b>                 | Une dos o más Strings y los devuelve concatenados en un nuevo String                    |

# Métodos del Objeto String

|                         |   |
|-------------------------|---|
| <b>endsWith(cadena)</b> | Comprueba si el String termina con los caracteres pasados por parámetro             |
| <b>charCodeAt()</b>     | Devuelve el unicode del carácter en el índice especificado                          |
| <b>includes(cadena)</b> | Comprueba si el String contiene la cadena pasada por parámetro                      |
| <b>localeCompare()</b>  | Comprueba si dos cadenas son equivalentes en la configuración regional actual.      |
| <b>repeat()</b>         | Devuelve un String con el número de copias de la cadena especificado por parámetro. |

# Objeto Date

- Los **objetos Date** representan en JavaScript un momento fijo en el **tiempo** en un formato independiente. El objeto Date contiene un Number que representa los milisegundos transcurridos desde el 1 de Enero de 1970 UTC.
- Para crear un nuevo objeto Date se lo **instancia** con **new Date()**



# Objeto Date

- Sin argumentos, crea un objeto Date para la fecha y la hora actuales:

```
let now = new Date();
alert( now ); // muestra en pantalla la fecha y la hora actuales
```

- Si se pasa un único argumento, y es de tipo string, entonces es analizado y convertido a fecha automáticamente.

```
let date = new Date("2017-01-26");
alert(date);
```



# Métodos del Objeto Date

Los objetos de la clase Date no tienen propiedades pero si **métodos**:

| MÉTODOS      | QUÉ HACE   |
|--------------|--|
| getDate()    | Devuelve el día del mes.   |
| getDay()     | Devuelve el día de la semana.  |
| getHours()   | Retorna la hora.   |
| getMinutes() | Devuelve los minutos.  |
| getMonth()   | Devuelve el mes (atención al mes que empieza por 0).   |
| getSeconds() | Devuelve los segundos.   |
| getTime()    | Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje. |

# Métodos del Objeto Date

Los objetos de la clase Date no tienen propiedades pero si métodos:

| MÉTODOS       | QUE HACE   |
|---------------|--|
| getFullYear() | Retorna el año con todos los dígitos. Usar este método para estar seguros de que funcionará todo bien en fechas posteriores al año 2000. |
| setDate()     | Actualiza el día del mes.  |
| setHours()    | Actualiza la hora.   |
| setMinutes()  | Cambia los minutos.  |
| setMonth()    | Cambia el mes (atención al mes que empieza por 0).   |
| setSeconds()  | Cambia los segundos.   |
| setTime()     | Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970.   |

# Objeto Math

- **Math** es un objeto incorporado que tiene **propiedades y métodos** para constantes y funciones **matemáticas**.
- Math funciona con el tipo **Number**. No funciona con **BigInt**.
- A diferencia de los demás objetos globales, el objeto Math no se puede editar. Todas las propiedades y métodos de Math son **estáticos**.



# Propiedades del Objeto Math

| PROPIEDADES  | QUE HACE                         |
|--------------|----------------------------------|
| Math.E       | Número de Euler                  |
| Math.LN2     | Equivalente a Math.log(2)        |
| Math.LN10    | Equivalente a Math.log(10)       |
| Math.LOG2E   | Equivalente a Math.log2(Math.E)  |
| Math.LOG10E  | Equivalente a Math.log10(Math.E) |
| Math.PI      | Número PI o $\pi$                |
| Math.SQRT1_2 | Equivalente a Math.sqrt(1/2).    |
| Math.SQRT2   | Equivalente a Math.sqrt(2).      |

# Métodos del Objeto Math

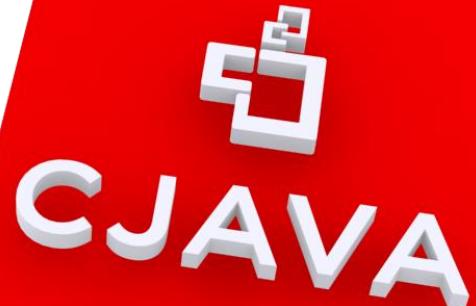
| MÉTODOS              | QUE HACE   |
|----------------------|--|
| Math.abs(x)          | Devuelve el valor absoluto de x.                               |
| Math.sign(x)         | Devuelve el signo del número: 1 positivo, -1 negativo          |
| Math.exp(x)          | Exponenciación. Devuelve el número e elevado a x.              |
| Math.expm1(x)        | Equivalente a Math.exp(x) - 1.                                 |
| Math.max(a, b, c...) | Devuelve el número más grande de los indicados por parámetro.  |
| Math.min(a, b, c...) | Devuelve el número más pequeño de los indicados por parámetro. |
| Math.pow(base, exp)  | Potenciación. Devuelve el número base elevado a exp.           |
| Math.sqrt(x)         | Devuelve la raíz cuadrada de x.                                |

# Métodos del Objeto Math

| MÉTODOS       | QUE HACE  |
|---------------|---|
| Math.cbrt(x)  | Devuelve la raíz cúbica de x.   |
| Math.random() | Devuelve un número al azar entre 0 y 1 con 16 decimales.              |
| Math.log(x)   | Devuelve el logaritmo natural en base e de x.                         |
| Math.log10(x) | Devuelve el logaritmo decimal (en base 10) de x.                      |
| Math.log2(x)  | Devuelve el logaritmo binario (en base 2) de x.                       |
| Math.log1p(x) | Devuelve el logaritmo natural de (1+x).                               |
| Math.round(x) | Devuelve x con <b>redondeo</b> ( <i>el entero más cercano</i> )       |
| Math.ceil(x)  | Devuelve x con <b>redondeo superior</b> ( <i>el entero más alto</i> ) |
| Math.floor(x) | Devuelve x con <b>redondeo inferior</b> ( <i>el entero más bajo</i> ) |
| Math.trunc(x) | Trunca el número x ( <i>devuelve sólo la parte entera</i> )           |

# Métodos del Objeto Math

| MÉTODOS            | QUE HACE   |
|--------------------|--|
| Math.sin(x)        | Seno de x  |
| Math.asin(x)       | Arcoseno de x                                    |
| Math.sinh(x)       | Seno hiperbólico de x                            |
| Math.cos(x)        | Coseno de x                                      |
| Math.acos(x)       | Arcocoseno de x                                  |
| Math.cosh(x)       | Coseno hiperbólico de x                          |
| Math.tan(x)        | Tangente de x                                    |
| Math.atan(x)       | Arcotangente de x                                |
| Math.tanh(x)       | Tangente hiperbólica de x                        |
| Math.hypot(a, b..) | Devuelve la raíz cuadrada de $a^2 + b^2 + \dots$ |



# JavaScript

EDWIN MARAVI

# Contenidos o temas

- Formulario web
- Controles básicos
- Controles de selección
- Eventos de controles y formularios

# Formulario web

- Un **formulario** es un conjunto de **controles** (botones, cajas de texto, casillas de verificación, botones radio, etc.) que permiten al usuario **introducir datos y enviarlos** al servidor web para su procesamiento.

# Formulario web

- La etiqueta que delimita un formulario es la etiqueta `<form> ...</form>`. Los atributos más importantes de la etiqueta `<form>` son:
  - **action:** contiene el nombre del agente que procesará los datos remitidos al servidor (por ejemplo, un script de PHP).
  - **method:** define la manera de enviar los datos al servidor. Los valores posibles son:
    - **get:** los valores enviados se añaden a la dirección indicada en el atributo action.
    - **post:** los valores se envían de forma separada.

# Formulario web

Ejemplo de **formulario HTML**:

Nombre:

Año de nacimiento:

Sexo:  Hombre  Mujer

```
<form action="ejemplo.php" method="get">
    <p>Nombre: <input type="text" name="nombre" size="40">
</p>
    <p>Año de nacimiento: <input type="number" name="nacido" min="1900"></p>
    <p>Sexo:
        <input type="radio" name="hm" value="h"> Hombre
        <input type="radio" name="hm" value="m"> Mujer
    </p>
    <p>
        <input type="submit" value="Enviar">
        <input type="reset" value="Borrar">
    </p>
</form>
```

# Controles básicos

**Cuadro de Texto <input type="text">**: Las cajas de texto de una sola línea se crean mediante una etiqueta **<input>** cuyo atributo **type** tiene el valor “**text**”.

```
<input type="text" name="texto">
```

Escriba algo:

# Controles básicos

**Cuadro de texto de Contraseña: <input type="password">:** Las cajas de texto de una sola línea específicas para contraseñas se crean mediante una etiqueta <input> cuyo atributo **type** tiene el valor “**password**”.

```
<input type="password" name="contrasena">
```

Contraseña:

# Controles básicos

**Cuadro de texto de Numérico: <input type="number">:** Las cajas de texto de una sola línea específicas para números se crean mediante una etiqueta **<input>** cuyo atributo **type** tiene el valor “**number**”. En principio, los valores admitidos por el control son números enteros.

Escriba un número:

```
<input type="number" name="numero">
```

Escriba un número:

Enviar

# Controles básicos

**Cuadro de texto de Fecha: <input type="date">**: Las cajas de texto de una sola línea específicas para fechas (días, meses, años) se crean mediante una etiqueta **<input>** cuyo atributo **type** tiene el valor “**date**”. El dato se envía con el formato AAAA-MM-DD donde AAAA es el número de año, MM el número de mes y DD el número de día.

```
Fecha: <input type="date" name="fecha">
```

Fecha: dd/mm/aaaa 

**Enviar**

# Controles básicos

**Cuadro de texto de Hora: <input type="time">:** Las cajas de texto de una sola línea específicas para tiempos (horas, minutos) se crean mediante una etiqueta **<input>** cuyo atributo **type** tiene el valor “**time**”. El dato se envía con el formato HH-MM donde HH son las horas y MM los minutos.

```
Hora: <input type="time" name="hora">
```

Hora:  ⏺

Enviar

# Controles básicos

- **Botones <input> y <button>:** Los botones se crean mediante la etiqueta <input> o mediante la etiqueta <button>. La diferencia entre ellos es que <input> sólo puede contener texto, mientras que <button> permite incluir elementos html como imágenes.
- El contenido del botón <input> se define mediante el atributo **value**, por lo que sólo puede contener texto. El contenido del botón <button> se escribe dentro del elemento, por lo que puede incluir **texto e imágenes**.

# Controles básicos

El **botón Submit** es el que permite al usuario remitir los datos al **servidor**. Se crea mediante una etiqueta **<input>** o **<button>** cuyo atributo **type** tiene el valor **submit**.

```
<input type="submit" value="Enviar">
```

Escriba algo:

```
<button type="submit">
  
  Enviar
</button>
```

Escriba algo:



# Controles básicos

**Botones reset <input> o <button>:** El botón Reset restablece los valores iniciales del formulario. Se crea mediante una etiqueta **<input>** o **<button>** cuyo atributo **type** tiene el valor **reset**.

```
<input type="reset" value="Borrar">
```

Caja de texto:

Casilla de verificación  
 Radio 1  Radio 2  Radio 3

**Borrar**

```
<button type="reset">  
    
  Borrar  
</button>
```

Caja de texto:

Casilla de verificación  
 Radio 1  Radio 2  Radio 3



**Borrar**

# Controles básicos

**Control Area de Texto <textarea>:** Las cajas de texto de varias líneas se crean mediante la etiqueta **<textarea>**. Los atributos obligatorios **rows** y **cols** establecen el número de filas y columnas iniciales de la caja, aunque los navegadores permiten modificarlo arrastrando la esquina inferior derecha.

```
<textarea name="texto" rows="4" cols="20"></textarea>
```



# Controles de Selección

**Botón de opción <input type="radio">:** Los botones radio se crean mediante una etiqueta <input> cuyo atributo **type** tiene el valor “**radio**”.

```
<input type="radio" name="boton" value="1">Opción 1
```

Opción 1

# Controles de Selección

Los **botones de radio** que tienen el mismo atributo **name** forman un **grupo**, es decir, que si se marca uno de ellos se desmarca automáticamente el resto.

```
<input type="radio" name="boton" value="1">Opción 1<br>
<input type="radio" name="boton" value="2">Opción 2<br>
<input type="radio" name="boton" value="3">Opción 3
```

- Opción 1
- Opción 2
- Opción 3

# Controles de Selección

**Casilla de Verificación <input type="checkbox">:** Las casillas de verificación se crean mediante una etiqueta **<input>** cuyo atributo **type** tiene el valor **"checkbox"**.

```
<input type="checkbox" name="casilla">Casilla 1
```

Casilla 1

# Controles de Selección

**Cuadro combinado <select>:** Los cuadros combinados se crean mediante la etiqueta **<select>**. Cada opción se define mediante la etiqueta **<option>**. El valor que se envía es el texto que aparece en la lista desplegable o menu, salvo si el elemento **<option>** contiene el atributo **value**.

```
<select name="menu">
  <option>Uno</option>
  <option value="2">Dos</option>
  <option value="three">Tres</option>
</select>
```

Elija una opción: Uno ▾

Enviar

# Controles de Selección

El atributo size permite definir la altura del control.

```
<select name="menu" size="3">
  <option value="1">Uno</option>
  <option value="2">Dos</option>
  <option value="3">Tres</option>
  <option value="4">Cuatro</option>
  <option value="5">Cinco</option>
</select>
```

Elija una opción:



Enviar



# CJAVA

siempre para apoyarte

Gracias