

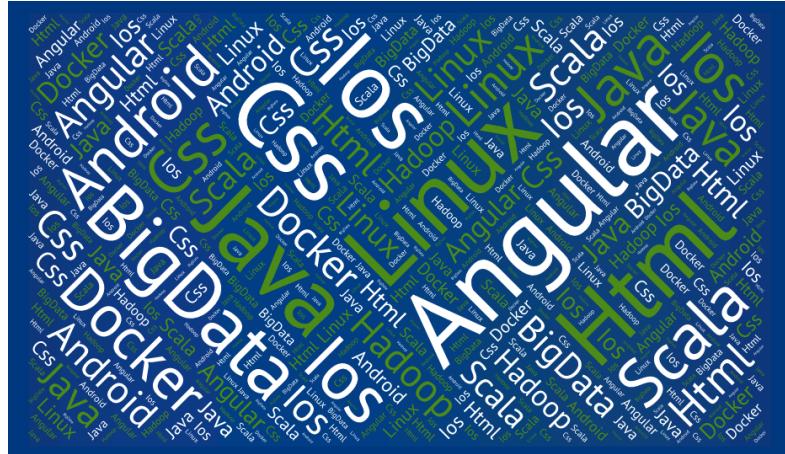


**FORMACIÓN EN NUEVAS TECNOLOGÍAS**

---

**JAVASCRIPT BÁSICO**

# ICONO TRAINING



## Formación en Nuevas Tecnologías



[www.iconotc.com](http://www.iconotc.com)



[linkedin.com/company/icono-training-consulting](https://linkedin.com/company/icono-training-consulting)



[training@iconotc.com](mailto:training@iconotc.com)

# FORMADORA



**Ana Isabel Vegas**



Consultora / formadora en Tecnologías  
de la Información.

¡Síguenos en las Redes Sociales!



# JAVASCRIPT BÁSICO



## DURACIÓN

↳ 20 horas



## MODALIDAD

↳ Remota en tiempo real.



## LUGAR/FECHAS /HORARIO:

↳ Días 28, 29, 30 de Noviembre y 1 de Diciembre de 2022. De 9:00 hs a 14:00 hs



## CONTENIDO:

1. Ciclo de vida de un programa JavaScript
2. Elementos y constructos esenciales del lenguaje
3. Programación Orientada a Objetos con JavaScript
4. Nociones de programación funcional en JavaScript
5. Colecciones de objetos
6. Los objetos estándar del navegador
7. Interacción de las páginas: el sistema de eventos
8. Gestión de formularios HTML
9. Biblioteca estándar de JavaScript
10. Introducción a AJAX

# 1. Ciclo de vida de un programa JavaScript

- a.Que es JavaScript
- b.Integración con HTML
- C.Otra forma de incluir código
- d.El navegador no soporta JavaScript

## 1.a. Qué es JavaScript .-



**JavaScript** es un lenguaje de programación para añadir interactividad a las páginas Web incrustándolo en el código HTML .



Nuestros programas en JavaScript no generan ningún tipo de código compilado, sino que este se interpreta en el navegador una vez que se descarga la página que lo contiene. A este tipo de lenguajes se les denomina **lenguajes interpretados**.



No necesitamos ninguna herramienta especial para programar en JavaScript. Simplemente usando el bloc de notas de Windows.



La mezcla de JavaScript con HTML se suele conocer con el nombre de **HTML Dinámico o DHTML**.

# 1.a. Qué es JavaScript .-



Algunos datos acerca del lenguaje JavaScript:

- Se llamaba originalmente LiveScript, pero Netscape cambió el nombre a JavaScript debido a que Netscape tenía soporte para Java (marketing).
- Es interpretado y orientado a objetos.
- Débilmente tipado (todas las variables son de tipo 'var').
- Utiliza una especificación de lenguaje llamada ECMAScript.
- Sintaxis similar a C, y convenciones de codificación similar a Java, aunque no están relacionados.
- Su mayor utilización es en el navegador (lado cliente), pero existe también por el lado servidor.
- Su uso se ha incrementado significativamente por el enfoque interactivo y dinámico de los sitios web actuales.

## 1.b. Integración con HTML .-

 Se utilizan las etiqueta contenedora <script>

 La estructura es la siguiente:

```
<script language="JavaScript">
    alert("Empezamos con JavaScript!!!!");
</script>
```

 Todas las líneas de código JavaScript se deben terminar con punto y coma (:).

 Normalmente la implementación se suele insertar entre las cabeceras <head> y </head>, y las llamadas a la función (ejecución) entre las etiquetas <body> y </body>.

## 1.c. Otra forma de incluir código .-

-  Otra forma de incluir código JavaScript en las páginas Web es a través del uso de archivos externos. Esto es de especial utilidad cuando queremos reutilizar un mismo código en varias páginas.
-  La sintaxis necesaria es la siguiente:

```
<script language="JavaScript" src="codigo.js"> </script>
```

# 1.d. El navegador no soporta JavaScript .-



Utilizamos la etiqueta contenedora <noscript>, que mostrará su contenido si el navegador no soporta lenguajes de guiones.

```
<script language="JavaScript">
    alert("Empezamos con JavaScript!!!!");
</script>
<noscript>
    con este navegador no podrá ver
    el contenido dinámico de mi página.
</noscript>
```

## 2. Elementos y constructores esenciales del lenguaje

- a. Comentarios
- b. Mayúsculas y minúsculas
- C. Variables
- d. Tipos de datos
- e. Ámbito de las variables
- f. Operadores
- g. Estructuras de control

## 2.a. Comentarios .-



Si el comentario ocupa una sola línea utilizamos barras inclinadas:

```
//Esto es un comentario de una línea
```



Si ocupa varias líneas lo comenzamos con /\*) y lo finalizamos con (\*/):

```
/* Esto es un comentario  
que me va a ocupar  
varias líneas */
```

## 2.b. Mayúsculas y minúsculas .-

-  JavaScript es un lenguaje sensible al uso de mayúsculas y minúsculas. Esto quiere decir que las diferencia por lo que hay que tener especial cuidado a la hora de definir variables.
-  No es lo mismo NUM que Num o num.

## 2.c. Declaración de variables .-

-  El nombre de un variable puede ser cualquiera, mientras no coincida con el de alguna palabra reservada del lenguaje.
-  Los nombres de variables siempre deben empezar por una **letra del alfabeto (mayúscula o minúscula)**, un **subrayado (\_)** o un **signo de dólar (\$)**, y sólo pueden constar de letras, números y subrayados. Por supuesto, no pueden incluirse espacios en el nombre de una variable.
-  Tenemos dos formas de declarar variables: **implícitamente y explícitamente**.

## 2.c. Declaración implícita .-



Consiste en escribir únicamente el nombre y asignarle un valor.

```
minombre = "Anabel";
```



Si la variable no se declara mediante el operador var, automáticamente se crea una variable global (aunque sea dentro de un método) con ese identificador y su valor.

## 2.c. Declaración explícita .-



Se usa la palabra reservada var seguida del nombre de la variable.

```
var minombre ;  
var minombre = "Anabel";
```



Podemos definir más de una variable en una misma sentencia si separamos sus nombres con coma

```
var variable1, variable2, variable3, variable4;
```

## 2.c. Ejemplo de variables.-

```
<html>

    <script language="JavaScript">

        var total;

        total = 5;

        alert(total);

    </script>

</html>
```

# Ejercicio variables.-

Crear un script con tres variables y que luego muestre el contenido de dichas variables en la pantalla.

## 2.d. Tipos de datos .-



En JavaScript tenemos 6 tipos primitivos:

- **String.** Cadenas de texto
- **Number.** Valores numéricos
- **Boolean.** Boléanos
- **Object.** Objetos
- **Null.** Objeto que todavía no existe
- **Undefined.** Indefinido, valor que presenta una variable definida antes de asignarle un valor.

## 2.d. Tipos de datos (continuación).-

- 💡 Cuando declaramos una variable, esta no pertenece a ningún tipo en concreto, tiene un valor indefinido (`Undefined`). Es cuando le asignamos un dato cuando pasa a ser de uno u otro tipo dependiendo del valor que albergue.
- 💡 Podemos averiguar el tipo de dato que contiene una variable utilizando la función `typeof`, que devuelve una cadena indicando dicha información.

```
var nombre = "Juan";
alert(typeof(nombre));
```

## 2.d. El tipo String o Cadena .-

- En JavaScript los textos se denotan por comillas dobles o comillas simples, pero no ambas a la vez.
- Dentro de una cadena delimitada por comillas dobles, podemos utilizar comillas simples y viceversa.

```
alert("Bienvenidos al 'curso' de JavaScript");  
  
alert( 'Bienvenidos al “curso” de JavaScript' );
```

## 2.d. Caracteres especiales .-



Cuando necesitemos utilizar un carácter especial dentro de una cadena, debemos emplear el signo de escape “\”, el cual actúa como carácter de control para incluir otros en una cadena. Los más comunes son:

Carácter	Significado
\n	Salto de línea
\r	Retorno de carro
\f	Salto de página
\t	Tabulador
\\"	El propio carácter “\”
\`	Comilla simple
\”	Comilla doble
\b	Carácter anterior

## 2.d. Ejemplo de caracteres especiales .-

```
var minombre = "Pepe";
var apellido1 = "López";
var apellido2 = "González";
alert(minombre + "\n" + apellido1 + "\n" + apellido2);
```

Si también le queremos insertar un tabulador:

```
alert(minombre + "\n" + "\t" + apellido1 + "\n" + apellido2);
```

O también :

```
alert(minombre + "\n\t" + apellido1 + "\n" + apellido2);
```

## 2.d. Concatenar texto .-



Se pueden concatenar varias cadenas de texto utilizando el operador “+”:

```
var minombre = "Nombre " + "Apellido";
```

**Resultado:**

```
minombre = "Nombre Apellido"
```

## 2.d. Tipos numéricos .-



En otros lenguajes existen multitud de tipos diferentes de números según su tamaño. En JavaScript podemos guardar indistintamente en una variable un número entero, uno en coma flotante del tamaño que sea, positivos, negativos..., sin preocuparnos por obtener un error porque la variable no lo admite.

```
var numnatural = 12345;  
var numentero = -382;  
var numcomaflotante = 3.14159;  
var numcomaflotante2 = -2.678753;  
var nummo1 = 6.023E+23;
```

## 2.d. Bases numéricas .-

-  JavaScript soporta tres bases numéricas diferentes cuando nos referimos a número enteros. Por omisión, se emplea la base decimal (base 10), pero también es posible utilizar las bases hexadecimal (base 16) u octal (base 8).
-  Para usar un numero hexadecimal debemos precederlo de 0x (cero x) y para números octales lo precedemos de 0 (cero).

```
var numhexadecimal = 0x54DE;  
var numoctal = 012345;
```

## 2.d. Ejemplo de bases numéricas.-

```
<html>  
<script language="JavaScript">  
    n = 0xFF32;  
    alert(n);  
    n=012345;  
    alert(n);  
    n=12345;  
    alert(n);  
</script>  
</html>
```

## 2.d. El tipo Booleano .-

-  Las variables booleanas o lógicas sólo pueden tomar dos valores posibles: true (verdadero) y false (falso). Dichos valores siempre son en minúsculas.
-  Generalmente, surgen como resultado de una comparación.
-  False equivale al número 0 y true equivale a cualquier número distinto de cero.

## 2.d. Valores especiales para variables numéricas .-



Existen unos cuantos valores específicos dentro de los números que indican situaciones especiales. Dichos valores solo están soportados para la versión 1.3 o superior de JavaScript.

Nombre	Significado
NaN	No es un número (Not a Number)
Infinity	Infinito. También puede ser menos infinito.

```
Ejemplo: if (isNaN (num)== false) {  
    alert ("si es un número");  
}
```

## 2.d. Tipo Nulo .-

- When a reference variable does not contain any value, its content is null.
- We will get this value when we define a variable and do not assign it an object, or when we explicitly assign it a null value to indicate that we want it to be empty.

```
var nombre;  
var nombre = null;
```

## 2.d. Ejercicio tipos de datos .-

Crear un script con una variable para cada tipo de datos.

La página mostrará el valor de dichas variables y el tipo de datos de contienen.

## 2.d. Conversión entre tipos de datos .-



Muchas veces nos vamos a encontrar con situaciones en las que tenemos un dato como numérico y desearíamos que fuese del tipo texto o viceversa.



En este caso deberíamos convertir el dato al tipo deseado antes de trabajar con el.



Tenemos dos tipos de conversión:



Conversión implícita



Conversión explícita

## 2.d.Conversión implícita .-



Qué ocurre cuando asignamos a una variable el resultado de operar con otras variables de distintos tipos?

→ Numérico y texto → el dato numérico pasa a ser un String si utilizamos un + sino lo convierte todo a numérico. Si la cadena no se puede transformar en número se obtiene un resultado indeterminado.

→ Booleano y número → el valor true será un 1 y false será 0.

## 2.d. Conversión explícita .-



Para ayudarnos a convertir una variable JavaScript proporciona las funciones: `parseFloat()`, `parseInt()` y `toString()`.

👉 `parseFloat()`; toma una cadena y la transforma en un número en coma flotante si ello es posible. En cuanto encuentra un carácter que no sea válido como número se detiene la conversión.

```
parseFloat("166.263"); → 166.263  
parseFloat("16FFCDE"); → 16  
parseFloat("FFCDE"); → NaN
```

## 2.d. Conversión explícita (continuación) .-

↳ `parseInt();` toma una cadena y la transforma en un número entero. Dispone de un argumento para escoger la base en la que consideramos que está el número contenido e la cadena de texto, y está comprendido entre 2 y 36. Si no se pone base se considera decimal. Recordamos que si iniciamos la cadena con `0x` se considera hexadecimal y si comienza por `0` será octal. Hay que tener cuidado con no poner números que comiencen por `0`.

```
parseInt("FFCDE",16);
```

## 2.d. Conversión explícita (continuación) .-



`toString();` coge un valor lo transforma en cadena de texto dependiendo del valor introducido:

Objeto	Resultado
Número	Devuelve una cadena con el numero
Booleano	Devuelve las cadenas true o false
Matriz	Tendremos una cadena con los elementos de la matriz transformados en cadenas y separados por comas.
Objeto	Devuelve la cadena [Object NombreObjeto], donde NombreObjeto es el nombre del tipo de objeto.
Función	Obtendremos una cadena con la definición de la función, así como su código.

En el caso de los números, tiene un argumento opcional para escoger la base en la que queremos que aparezcan representados en la cadena.

```
var1 = (13).toString(16);
var2 = (13).toString(2);
alert(Var1 + "\n" + Var2);
```

# Ejercicio de conversiones.-

-  Hacer un script que defina una variable con el dato “1289.45” transformarlo a numero flotante y luego a numero entero (en dos variables diferentes).
-  Después definir otras dos variables una con un dato numérico y otra con un dato booleano y convertir ambas a cadenas de texto.
-  Mostrar el contenido de todas las variables.

## 2.e. Ámbito de las variables .-



Dependiendo de donde declaremos las variables nos vamos a encontrar con dos tipos:



**Variables de ámbito global**



**Variables de ámbito local**



Es importante saber cual es el ámbito de mi variable porque dependiendo del programa producen resultados diferentes.

## 2.e. Variables globales .-

-  Una variable es Global cuando se puede acceder a su contenido desde cualquier punto de la página donde reside el script.
-  Se tiene que declarar fuera de cualquier procedimiento o función definida por el usuario.
-  Permanecen en memoria mientras esté cargada la página donde fueron declaradas.
-  Sólo se deben utilizar cuando realmente exista la necesidad de compartir información entre todas las partes de código de una página, puesto que de otro modo se estaría desperdiciando memoria innecesariamente.

## 2.e. Ejemplo de variables globales .-

```
<script language="JavaScript">
    var total=0;
    function calculasuma(a,b) {
        total = a+b;
    }
</script>

calculasuma(12,3);
```

## 2.e. Variables locales .-

-  Una variable es local cuando se define en el entorno de una función, dejando de existir al terminar de ejecutarse ésta.
-  Cuando la función termina, esa variable o variables serán desalojadas de la memoria por el motor del lenguaje, liberando toda referencia a las mismas.
-  Por eso reciben el nombre de variables locales, ya que su ámbito de validez está restringido a una determinada zona del código de Script.

## 2.e. Ejemplo de variables locales .-

```
<script language="JavaScript">
    function calculasuma(a,b) {
        var total=0;
        total = a+b;
    }
</script>

calculasuma(12,3);
```

## 2.e. Ejemplo de ambas variables .-

```
<script language="JavaScript">
    var total = 0;

    function f1() {
        var total;
        total = 5;
        alert(total);
        f2();
    }

    function f2() {
        alert(total);
    }

    f1();
</script>
```

## 2.f. Operadores aritméticos.-

Operación	Operador
Suma y resta	+,-
Multiplicación y división	* , /
Cambio de signo	-
Incremento	++
Decremento	--
División entera	~~ (num1/num2)
Resto de división (módulo)	%

## 2.f. Operadores lógicos .-

Operación	Operador
AND	&&
OR	
NOT	!



AND : Sólo devuelve true si los operandos son ciertos.



OR : Devuelve true si alguno de los operandos es cierto.



NOT : Devuelve el valor contrario de los operandos.

## 2.f. Operadores Relacionales .-

Operación	Operador
Igual a	<code>= =</code>
Distinto de	<code>!=</code>
Mayor, mayor o igual	<code>&gt;,&gt;=</code>
Menor, menor o igual	<code>&lt;,&lt;=</code>
Igualdad estricta	<code>= = =</code>
Distinto estrictamente	<code>!= =</code>



Las comprobaciones estrictas comparan además del dato el tipo de las variables. Solo devolverán true si tienen el mismo dato y además son del mismo tipo.

## 2.f. Operadores Bit a bit .-

Operación	Operador
AND	&
OR	
XOR	^
NOT	~
Rotación a la izquierda	<<
Rotación a la derecha	>>
Rotación a la derecha sin signo (Positivo 0 y negativo 1)	>>>



- AND; Devuelve 1 si ambos son 1.
- OR; Devuelve 1 si alguno son 1.
- XOR; Devuelve 1 si uno de ellos es 1 y el otro 0.
- NOT; Devuelve el valor contrario.

## 2.f. Operadores de asignación

Operación	Operador
Asignación directa	=
Con adición o concatenación	+=
Con sustracción	-=
Con multiplicación	*=
Con división	/=
Con módulo	%=
Con operaciones bit a bit	&=,  =, ^=, <<=, >>=, >>>=

## 2.f. Otros operadores .-

Operación	Operador
Concatenación de cadenas	+
Condicional	?:
Acceso a miembro	.
Coma (separador)	,
Creación de un objeto	new
Borrado de elementos	delete
Tipo de objeto	typeof
Obviar valor de expresión	void

## 2.f. Orden de ejecución (Precedencia)

Operadores	Precedencia
. () []	1
++ -- ~ ! typeof new delete void	2
* / %	3
+ -	4
<< >> >>>	5
< <= > >=	6
== != === !==	7
&	8
^	9
	10
&&	11
	12
?:	13
Asignación	14

## 2.f. Incrementos y decrementos

-  `a = a + 1;` es lo mismo que `a++;`
-  A esta forma de utilizar el operador se le llama postincremento. Su acción es aumentar el valor de la variable y luego devolver dicho valor. También se puede utilizar precediendo al nombre de la variable, será un preincremento.
-  Lo mismo ocurre con el operador `--` para decrementos. Si se coloca detrás de la variable, primero devuelve el valor de ésta, y después lo disminuye en la unidad. Si lo colocamos delante, entonces se disminuye y luego devuelve el valor.

```
var n1 = 3;  
var n2, n3;  
n2 = n1--;  
n3 = --n1;  
alert(n1 + "\n" + n2 + "\n" + n3);
```

## 2.g. Estructuras de control .-

↳ If-else

↳ Switch - case

↳ For

↳ For - in

↳ While

↳ Do - while

## 2.g. If - else .-

 Permite tomar decisiones en función de unas determinadas condiciones que se impongan para ejecutar uno u otro código según lo que indiquen dichas condiciones. Su sintaxis es la siguiente:

```
if (condición) {  
    código si cierto;  
} else {  
    código si falso;  
}
```

 Comprueba la validez o falsedad de una condición. Si esta condición es verdadera se ejecuta el código contenido entre las primeras llaves. Si es falsa, se interpreta el código tras las segundas llaves.

## 2.g. If - else (continuación).-



La cláusula else no es obligatoria. Si la condición fuese falsa no se llevaría a cabo ninguna acción. Además si no ponemos else y el código de las primeras llaves es tan solo de una línea podemos utilizar este código:

```
if (condición)      sentencia;  
if (condición)      sentencia1, sentencia 2;
```

# Ejercicio .-

-  Hacer un formulario donde se pida el nombre al usuario, poner el botón de Enviar.
-  Al hacer clic en el botón que se compruebe que el campo de texto nombre no este vacío. Si lo está que salga un mensaje avisándolo.

# Ejercicio .-



Repetir el ejercicio anterior pero esta vez añadiremos un campo de apellidos.



Al hacer la comprobación pediremos el nombre si no se ha introducido y el apellido si no está.

## 2.g. Switch - case .-



Durante la ejecución de un programa puede darse el caso de necesitar tomar una decisión a partir de múltiples posibilidades, pero donde sólo una de ellas va a ser posible en cada ocasión. Sólo está disponible a partir de la versión 1.3 de JavaScript y la 3 de JScript. Su sintaxis es:

```
switch (expresion) {  
    case valor1:  
        codigo1;  
        break;  
    case valor2:  
        codigo 2;  
        break;  
    default:  
        codigo por omisión;  
}
```

# Ejercicio .-



Hacer un formulario con botones de opción donde se pregunte por un rango de edades.



Al hacer clic en una opción se debe llamar a una función que guarde el valor en una variable.



Dependiendo de la edad elegida mostrar un mensaje en la pantalla.

## 2.g. For .-

-  Un bucle es aquella operación o conjunto de operaciones que se repite cierto número de veces hasta llegar a un estado que le obliga a detenerse, permitiendo que continúe la ejecución normal hacia delante del código. A las operaciones que se repiten en cada vuelta se las denomina conjuntamente Cuerpo del bucle.
-  Cuando se sabe de antemano el número de veces que se va a repetir el cuerpo del bucle antes de detenerse, se dice que el bucle es determinado. Si no se pude saber de antemano cuándo se detendrá, se dice que es un bucle indeterminado.

## 2.g. For (continuación).-



La sintaxis es la siguiente:

```
for (contador=valorinicial; condición del bucle; incremento) {  
    cuerpo del bucle;  
}
```



La variable contador se declara implícitamente y sólo es válida dentro del cuerpo del bucle.



La condición de bucle es una expresión que indica si el cuerpo del bucle se debe ejecutar una vez más o no. Mientras esta condición sea verdadera se repetirá el bucle. En el momento en que sea false se detendrá. Si inicialmente ya es falsa, el cuerpo del bucle no se ejecutará nunca.



Los bucles for se pueden anidar.

## 2.g. For - in .-



Recorre todos los elementos de un array. Consiste en ejecutar el cuerpo del bucle tantas veces como miembros tenga una colección de objetos o una matriz.

```
for (indice in variable_array) {  
    sentencias;  
}
```

## 2.g. While .-



Itera mientras la condición de final se cumpla. Comprueba la condición al principio. Si es falsa no se ejecuta

```
while (condición) {
    cuerpo del bucle;
}
```

## 2.g. Do - while .-



Itera mientras la condición de final se cumpla. Comprueba después, al menos se ejecuta una vez.

```
do {  
    cuerpo del bucle;  
}  
while (condición);
```

## 2.g. Instrucciones break y continue .-

-  Con la instrucción break salimos del bucle sin necesidad de que se cumpla su condición de finalización.
-  Con la instrucción continue hacemos que el bucle ejecute la siguiente vuelta sin necesidad de llegar al final del código de su cuerpo.

# Ejemplo .-

```
<script language="JavaScript">
    function NoMult5(n1, n2) {
        var res = "";
        for (i=n1; i<=n2; i++) {
            if (i % 5 == 0) {
                continue;
            }
            res += i + " ";
        }
        return res;
    }

    alert (NoMult5(2, 127));
</script>
```



Ejemplo que muestra los números en un rango dado menos los múltiplos de 5



Como vemos si el número es divisible por 5, se salta a la vuelta siguiente sin pasar por la línea que concatena los números en el resultado.

### 3. Programación orientada a objetos con JavaScript

a. El objeto Math

b. El objeto String

C. El objeto Date

## 3.a. Operaciones matemáticas. El objeto Math.-



En JavaScript para utilizar una función matemática debemos invocar antes al objeto Math.



Dicho objeto contiene las siguientes funciones:

↳ log

↳ exp

↳ sqrt

↳ pow

↳ Abs

↳ floor

↳ ceil

↳ round

↳ Random

↳ Cos

↳ Max

↳ Min

↳ with

## 3.a. Funciones logarítmicas y exponenciales .-



Las funciones disponibles con métodos de Math son log (logaritmo neperiano o en base ‘e’ ) y exp (elevar el numero ‘e’ a un parámetro).

```
var logaritmo = Math.log(1000); → 6,907755....  
var e4 = Math.exp(4); → e4
```

## 3.a. Raíces cuadradas .-



La función es sqrt y su sintaxis es la siguiente:

```
var raiz;  
raiz = Math.sqrt(9);
```

## 3.a. Potencias de números .-



La función es pow y su sintaxis será:

```
Math.pow(3,2); → 32
```

## 3.a. Funciones de acotación y redondeo .-



La función abs devuelve el valor absoluto de un número dado.

**Math.abs(-3.9832); → 3.9832**



La función floor devuelve el número entero más cercano al que se le pase como parámetro y que además sea menor que éste.

**Math.floor(3.9832); → 3**  
**Math.floor(-3.9832); → -4**



La función ceil devuelve el número entero más cercano al que se le pase como parámetro, pero que sea mayor que éste.

**Math.ceil(3.1832); → 4**  
**Math.ceil(-3.1832); → -3**

### 3.a. Funciones de acotación y redondeo (cont.).-



La función round redondea el valor de un número para ajustarlo al número entero que este más próximo.

Si el número pasado como parámetro tiene una parte decimal mayor o igual que 0,5 se obtiene el siguiente número entero.

**Math.round(3.1832); → 3**

**Math.round(3.8832); → 4**

**Math.round(3.5); → 4**



Lamentablemente, no existe ninguna función que nos permita redondear un decimal a la cantidad de cifras decimales que deseemos.

## 3.a. Números aleatorios .-



Para esta tarea, JavaScript proporciona el método random del objeto Math. Este se encarga de generar una secuencia de números pseudo-aleatorios a partir de un valor llamado semilla. Siempre devuelve una valor entre 0 y 1.

```
Math.random();
```

## 3.a. Funciones trigonométricas .-



Las funciones trigonométricas proporcionadas por JavaScript a partir del objeto Math son el seno (sin), coseno (cos), tangente(tan), arcocoseno (acos), arcoseno (asin), arcotangente (atan) y arcotangente a partir de los lados del ángulo (atan2).

```
var coseno = Math.cos (Math.PI/2);
```



El inconveniente es que el ángulo que se les pasa como argumento para el cálculo debe estar expresado en radianes y no en grados sexagesimales. Debido a esto si queremos pasar grados sexagesimales a alguna de estas funciones, se deben multiplicar antes por el factor pi/180. Además, como las funciones inversas (atan, acos, asin y atan2) devuelven un ángulo en radianes, para obtener el correspondiente en grados se debe multiplicar por 180/pi. Recordamos que  $2 * \pi$  radianes equivalen a  $360^\circ$ .

## 3.a. Funciones máximo y mínimo .-



Las funciones max y min se encargan de determinar el máximo y el mínimo respectivamente.

```
Math.max(22,95);    →    95  
Math.min(22,95);    →    22
```

## 3.a. La cláusula with .-



Para no tener que estar constantemente escribiendo la palabra Math podemos utilizar la cláusula with.

```
with (Math) {  
    res = (exp(x) - exp(x)) * sin(y) / PI * log(z);  
}
```

en vez de escribir:

```
res = (Math.exp(x) - Math.exp(x)) * Math.sin(y) / Math.PI * Math.log(z);
```

## 3.b. Funciones del objeto String .-

 El objeto String recoge todas las funciones para trabajar con cadenas de texto.

 Funciones principales:

 `fromCharCode`

 `slice`

 `charCodeAt`

 `substring`

 `indexOf`

 `split`

 `lastIndexOf`

 `toLowerCase`

 `substr`

 `toUpperCase`

### 3.b. La función fromCharCode .-



Toma como argumento uno o más números separados por comas, y devuelve una cadena de texto con los caracteres Unicode asociados a dichos números.

```
var sPrueba = String.fromCharCode (74,  
97, 118, 97);  
alert (sPrueba);
```

## 3.b. La función charCodeAt .-

-  Es el proceso inverso al anterior. En este caso se pasa la cadena de texto y devuelve el código Unicode asociado a la letra que se encuentre en la posición especificada.

```
var sPrueba = “Java”;  
alert (sPrueba.charCodeAt(3));
```

-  Hay que tener en cuenta que empieza contando en la posición 0.

## 3.b. La función indexOf .-



Devuelve la posición dentro de una cadena donde se dé la primera ocurrencia de la cadena que se está buscando. Además, se comenzará a buscar desde la posición especificada opcionalmente como segundo parámetro. Si no se utiliza dicho parámetro opcional para el carácter a partir del cual se debe comenzar a buscar, la búsqueda se iniciará desde el primer carácter de la cadena. La sintaxis es:

```
Cadena_donde_se_busca.indexOf(cadena_a_buscar, [posición_inicial]);
```

```
email.indexOf("@");
```



Distingue entre mayúsculas y minúsculas.

## 3.b. La función lastIndexOf .-

- Funciona exactamente igual que indexOf pero realiza la búsqueda de derecha a izquierda en lugar de izquierda a derecha.

## 3.b. La función substr .-



Toma como parámetros la posición de comienzo de la subcadena a extraer y la longitud de la misma.

```
alert("Esto es JavaScript".substr(3, 6));
```

devolvería

o es J



Si el segundo parámetro se omite, devuelve el fragmento de cadena comprendido entre la posición dada y el final de la misma.

## 3.b. La función slice .-



Es muy similar a la anterior. La diferencia es que en el segundo parámetro pasamos la posición del último carácter a extraer. El ultimo no está incluido.

```
alert("Esto es JavaScript".slice(3, 6));
```

devolvería o e

## 3.b. La función substring .-



Es igual a la anterior pero en este caso el segundo parámetro no es opcional.

```
alert("Esto es JavaScript".substring(3, 5));
```

## 3.b. La función split .-

 Coge una cadena de texto y la separa en diferentes partes atendiendo a un carácter separador que se pasa como parámetro. El resultado se almacena en una matriz.

 Es muy útil cuando cogemos una cadena con el nombre y los apellidos y lo queremos separar en tres partes para almacenarla en una base de datos en diferentes campos (nombres, apellido1 y apellido2).

```
var nombre;  
nombre = "Pepe López Sáez".split(" ");
```

## 3.b. La función toLowerCase .-



Convierte una cadena de texto a minúsculas.

```
"Esto es JAVASCRIPT".toLowerCase();
```

## 3.b. La función toUpperCase .-



Convierte una cadena de texto a mayúsculas.

```
"Esto es JAVASCRIPT".toUpperCase();
```

### 3.c. Fechas y Horas .-

 Una variable que contenga una fecha contiene realmente un valor numérico que expresa el número de milisegundos transcurridos desde el 1/1/1979 a las 00:00:00 horas.

### 3.c. Constructores del objeto Date .-



Para obtener la fecha y hora actuales en una variable, basta con definir una nueva instancia de Date:

```
var hoy = new Date( );
```



mostrará una cadena del tipo: “Tue Dec 28 11:13:56 UTC +0100 1999-12-28”

### 3.c. Constructores del objeto Date (cont.).-



Existen varios constructores:

```
var fecha = new Date( "mes, día, año horas:minutos:segundos " )
```

```
var fecha = new Date(año, mes, día [horas, [minutos, [segundos, [milisegundos]]]])
```

```
var fecha = new Date("May, 23, 1972 21:00")  
var fecha = new Date(1972, 4, 23, 21, 0)
```



En el segundo caso el mes se cuenta a partir de 0 (enero) hasta 11 (diciembre). El año se debe escribir con cuatro cifras, también podría ser en dos pero indicaría un año del siglo XX.

## 3.c. Métodos del objeto Date .-



Todas las funciones para poder trabajar con fechas las encontramos dentro del objeto Date:



Algunas de ellas son:

↳ `getTime`

↳ `getTimezoneOffset`

↳ `getDay`

↳ `getDate`

↳ `getMonth`

↳ `getYear`

↳ `getFullYear`

↳ `getHours`

↳ `getMinutes`

↳ `getSeconds`

↳ `getMilliseconds`

### 3.c. El método `getTime` .-



Obtiene solamente la parte correspondiente a la hora en milisegundos, rechazando la fecha.

```
var ahora = new Date();
document.write(ahora.getTime());
```

### 3.c. El método `getTimezoneOffset()` .-

 Nos da la diferencia horaria en minutos del ordenador con respecto al meridiano de Greenwich. El valor depende por tanto de la zona o huso horario para el que esté configurado el ordenador, pudiendo ser negativo o positivo según esté en la zona oriental u occidental. Lo devuelve en minutos

```
var ahora = new Date();
document.write(ahora.getTimezoneOffset());
```

### 3.c. El método getDay .-



Nos dice en el día de la semana que estamos, empezando a contar en domingo (0) hasta el sábado (6).

```
var fecha = new Date();
document.write("Hoy es "+fecha.getDay());
```

## 3.c. El método getDate .-



Obtiene el día del mes dentro de un fecha dada.

```
var fecha = new Date();
document.write("Hoy es día: "+fecha.getDate());
```

### 3.c. El método getMonth .-



Devuelve el número correspondiente al mes contenido en el objeto Date. Se comienza a contar en 0, que correspondería al mes de enero.

```
var fecha = new Date();
document.write("Este mes es el "+fecha.getMonth() );
```

### 3.c. El método getYear .-



Devuelve el año contenido en la variable de tipo fecha. Entre 1900 y 1999 sólo devuelve las dos últimas cifras, para el resto de fechas devuelve los cuatro dígitos.

```
var fecha = new Date();
document.write("Este año es el "+fecha.getYear());
```

### 3.c. El método getFullYear .-



Igual que el anterior, solo que en este caso siempre se devuelven las cuatro cifras.

```
var fecha = new Date();
document.write("El año actual es "+fecha.getFullYear());
```

### 3.c. El método getHours .-



Permite obtener el valor de la hora en una variable Date, sin incluir la fecha actual, ni los minutos ni los segundos.

```
var fecha = new Date();
document.write("Son las "+fecha.getHours()+" horas.");
```

## 3.c. El método getMinutes .-



Nos devuelve los minutos de la hora dada.

```
var fecha = new Date();
document.write("Son las "+fecha.getHours());
document.write(": " + fecha.getMinutes());
document.write(": " + fecha.getSeconds());
```

### 3.c. El método getSeconds .-



Permite obtener la parte correspondiente a los segundos en una variable de hora.

```
document.write("Son las "+fecha.getHours() );
document.write(": " + fecha.getMinutes() );
document.write(": " + fecha.getSeconds() );
```

### 3.c. El método getMilliseconds .-



Facilita el número de milisegundos después del segundo dentro de una variable de tipo Date.

```
var fecha = new Date();
document.write("Son las "+fecha.getHours());
document.write(": " + fecha.getMinutes());
document.write(": " + fecha.getSeconds());
document.write(": " + fecha.getMilliseconds());
```

# Fundamentos POO

# Características POO



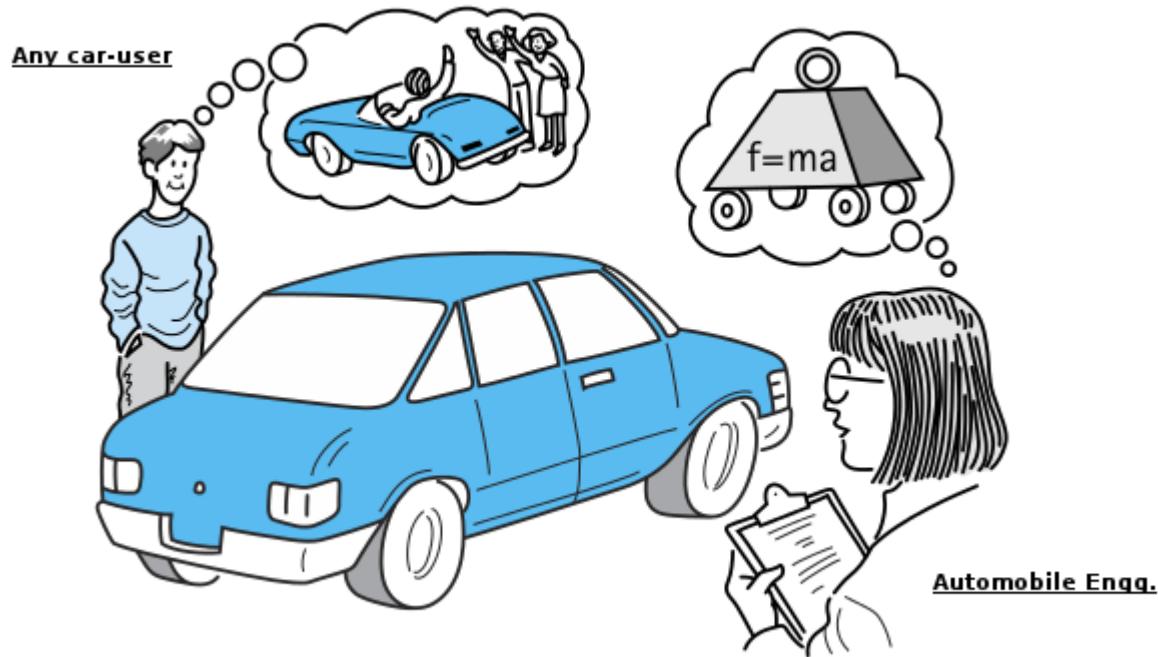
Existen cuatro conceptos fundamentales dentro de la Programación Orientada a Objetos que se relacionan entre sí y que nos permitirán tener las riendas de nuestro código:

- **Abstracción:** proceso mental de extracción de las características esenciales de algo, ignorando los detalles superfluos.
- **Encapsulación:** proceso por el que se ocultan los detalles del soporte de las características esenciales de una abstracción.
- **Modularización:** proceso de descomposición de un sistema en un conjunto de módulos o piezas independientes y cohesivos (con significado propio). Lo adecuado es conseguir los mínimos acoplamientos.
- **Herencia:** proceso de estructuración por el que se produce una organización (jerarquía) de un conjunto de elementos en grados o niveles de responsabilidad, incumbencia o composición entre otros.
- **Polimorfismo:** proceso para ver el objeto de diferentes modos

# Abstracción



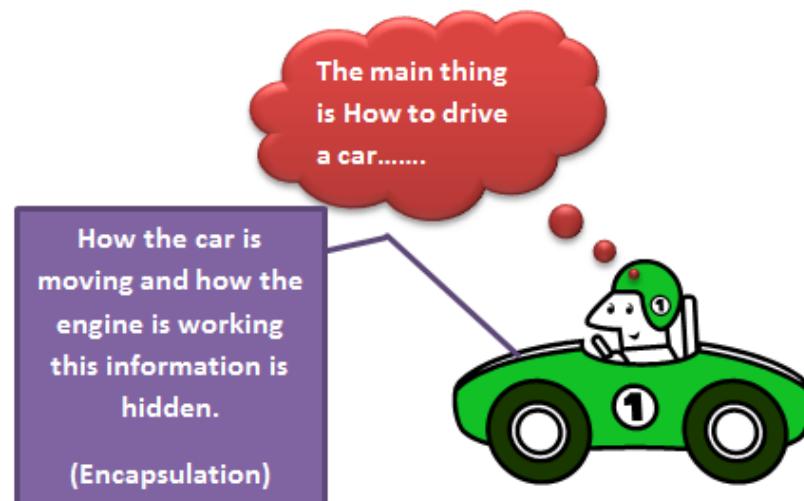
Consiste en **aislar** por supresión intencionada, u ocultamiento, **las cualidades de un objeto** para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.



*An abstraction includes the essential details relative to the perspective of the viewer*

# Encapsulamiento

- Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de componentes del sistema.
- Técnica para la ocultación de información que ayuda a definir mejor los módulos.
- Los módulos deben diseñarse para que la información que manejan está oculta a los otros módulos, de tal forma que entre módulos intercambien el volumen mínimo posible de información.



# Modularidad

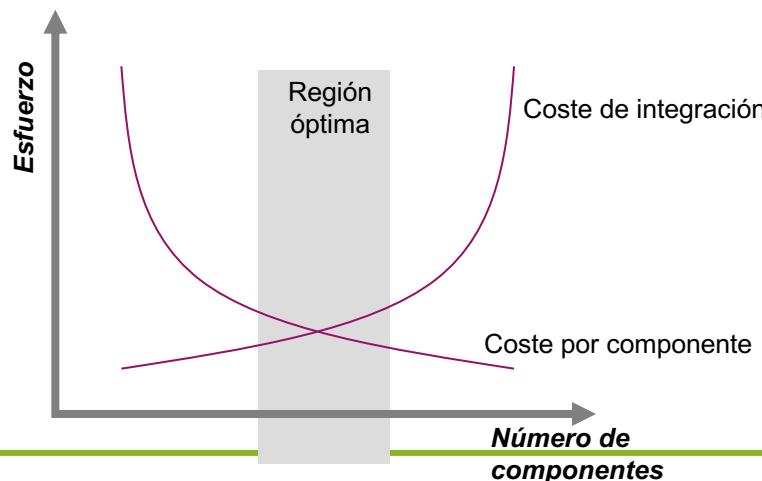


Consiste en dividir el software en diferentes componentes, resolver cada uno por separado y por último unirlos en una solución única.



Ante un problema complejo es más fácil resolverlo cuando se divide en piezas manejables.

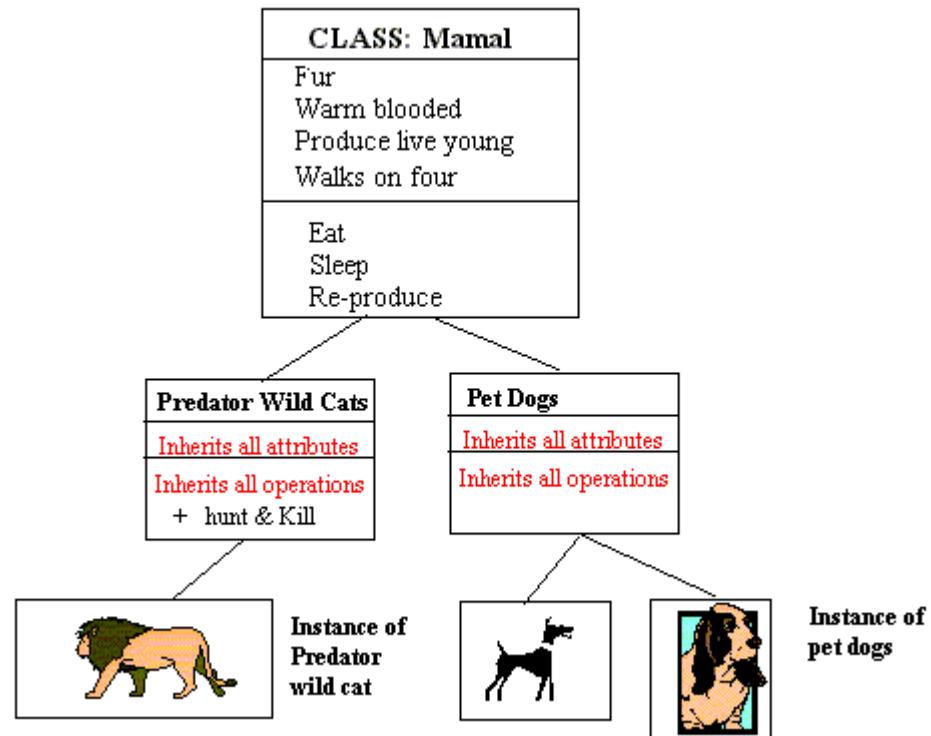
**¿Cuál es el número óptimo de componentes en el que debo dividir un programa?**



- Existe una región óptima en la que tanto el esfuerzo de integración como la dificultad de cada componente es baja.

# Herencia

- Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación.
- Los objetos heredan las propiedades, relaciones y los comportamientos de todas las clases a las que pertenecen.
- La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo compartir y extender las propiedades y el comportamiento sin tener que volver a implementarlo.



# Polimorfismo



Permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento.



Ben ten

Tutorial4us

Transform Form of Ben ten



# Objetos



Un **objeto** es un recurso dinámico que se crea y se almacena en memoria durante un tiempo determinado. A los objetos también se les conoce con el nombre de **instancias** de clase porque realmente esto es lo que son, una copia de la clase con unos valores determinados.

```
var alumno1 = new Alumno("Juan", "Lopez", 7.5);
var alumno2 = new Alumno("Maria", "Arias", 5.6);
```

# Clases



Una **clase** la podríamos definir como la plantilla a partir de la cual se va a generar el objeto.

```
class Alumno{
```

```
}
```

# Propiedades



Las propiedades o también llamadas atributos o campos recogen las características de la clase.



Una propiedad se entiende como una variable global a la cual se puede acceder desde cualquier otro recurso dentro de la misma clase y dependiendo del modo de acceso, también se podrá acceder a ella desde otras clases.

```
class Alumno{  
    nombre;  
    apellido;  
    nota;
```

# Métodos

- Utilizamos los métodos para introducir el código de nuestra aplicación. Podremos invocar a un método tantas veces como se quiera por lo cual nos permite una reutilización de código para aquellas tareas repetitivas.
- Igual que ocurre con las propiedades, un método se puede invocar desde otro recurso declarado en la misma clase y dependiendo de su acceso, se podrá llamar desde otras clases.

```
mostrarInformacion(){  
    return "NOMBRE :: " + this.nombre +  
        " | APELLIDO :: " + this.apellido +  
        " | NOTA :: " + this.nota;  
}
```

# 4. Nociónes de programación funcional en JavaScript

- a. Definición de funciones
- b. Funciones anónimas

## 4.a. Definición de funciones .-

-  Las funciones nos permiten conseguir la reutilización de código. Una función es, básicamente, un fragmento de código al que otorgamos un nombre y que podemos ejecutar siempre que lo necesitemos con sólo invocarlo a través de dicho nombre.
-  Para definir una función en JavaScript, se emplea la palabra reservada `function` (siempre en minúsculas) seguida de un identificador, que obedece las mismas reglas que los nombres de las variables.

```
function Mifucion() {  
    alert ("Hola");  
}  
  
Mifucion();
```

## 4.a. Definición de funciones (continuación) .-

- En las funciones, el paréntesis se utiliza para albergar la lista de argumentos (o parámetros) que definen su funcionamiento.
- Cuando una función necesita devolver algún valor u objeto como resultado de una ejecución, se debe emplear la palabra reservada return para hacerlo.

```
function calculasuma(a,b) {  
    return (a+b);  
}  
  
alert(calculasuma(12,3));
```

- Si una función no devuelve valor alguno, se le denomina método. No tienen la expresión return en su interior.

## 4.b. Funciones anónimas .-



Este tipo de función es ideal para el caso que se necesita definir una función sencilla que sólo se utilizará una vez y también para funciones que no necesitan un nombre.

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

# Programación funcional

# Definición

- El nombre ya lo sugiere: la programación funcional o functional programming se centra en las **funciones**.
- En un programa funcional, todos los elementos pueden entenderse como funciones y el código puede ejecutarse mediante **llamadas de función secuenciales**.

```
var nombres = ["Pepe", "Juan", "Antonio", "Laura", "Sofia", "Ismael"];

for(var i=0; i<nombres.length; i++){
    console.log(nombres[i]);
}

// Aplicando programacion funcional
nombres.forEach(nombre => {
    console.log(nombre);
});
```

# Funciones principales

- map:** El resultado conserva la forma del argumento de entrada, probablemente con diferente tipo.
- filter:** El resultado tendrá la misma forma que el argumento de entrada, probablemente de menor longitud.
- reduce:** El resultado puede quedar totalmente transformado.

```
numeros = numeros.map(num => num * 2);
console.log(numeros);

// Filtrar elementos de una colección
var mayores = numeros.filter(num => num > 5);
console.log(mayores);

var resultado = [5,9,2,6].reduce((total, num) => total + num);
console.log(resultado);
```

# Principio de inmutabilidad



Todo proceso que se lleva a cabo dentro de la función no modifica los valores iniciales, ya que se hace sobre una copia.

```
// El array no cambia, solo se aplican las modificaciones en una copia  
// se le llama principio de inmutabilidad.  
numeros.map(num => num * 2);  
console.log(numeros);
```

# Combinar funciones



Podemos encadenar funciones:

```
var resultado3 = ["5","9","2","6"]
    .map(dato => parseInt(dato))
    .filter(num => num > 5)
    .reduce((total, num) => total + num);
```



En este ejemplo primero se convierte cada elemento de tipo texto a un numero entero, luego se filtran los números mayores que 5 y por ultimo se calcula su suma. Mostrando por ultimo su valor por consola:

```
console.log(resultado3);
```

# 5. Colección de objetos

- a. Matrices
- b. Tipos de matrices
- C. Definición de matrices
- d. Acceso a los elementos
- e. Matrices multidimensionales
- f. Acceso a matrices multidimensionales
- g. Métodos del objeto Array

## 5.a. Matrices .-

-  Una matriz es un conjunto de elementos colocados de forma adyacente en la memoria de manera que nos podemos referir a ellos con un solo nombre común mientras que, por otro lado, no se pierde la independencia de los mismos.
-  Es un modo de agrupar datos para que se guarden ordenadamente y sean más cómodos de manejar y gestionar. Todo elemento pertenece a una matriz lleva asociado un índice de forma única: siempre se puede acceder a un elemento determinado e cualquier matriz si se conoce su índice.

## 5.b. Tipos de Matrices .-

-  Las matrices se pueden clasificar según dos puntos de vista:
-  Según la estabilidad en su tamaño:
  -  Si la matriz tiene un tamaño fijo, se dice que es una **matriz estática**
  -  Si la matriz puede cambiar su tamaño para adaptarse a las condiciones del código, se dice que es una **matriz dinámica**.
-  Según sus dimensiones:
  -  Si la matriz consiste tan sólo en un grupo de elementos que quedan determinados mediante un único número o índice, se denominan **Listas** o mas comúnmente **Arrays**. Son matrices de una sola dimensión.
  -  Si la matriz precisa de dos o más índices para determinar cualquiera de sus elementos se llama **matriz multidimensional** o simplemente **matriz**

## 5.c.Definición de matrices .-

- En JavaScript tenemos tres formas de definir matrices:
- Consiste en indicar simplemente que deseamos que una variable contenga una matriz, sin especificar su tamaño ni sus elementos. Dado que en JavaScript las matrices se representan mediante un objeto llamado Array, lo que haremos será crear un nuevo ejemplar del mismo usando el operador new, el cual sirve para crear instancias de objetos. Por lo cual la sintaxis será:

```
var miarray = new Array();
```

## 5.c. Definición de matrices (continuación).-



Cuando conocemos de antemano el número de elementos que queremos que vayan en la matriz, podemos definir ésta directamente diciendo cuántos serán:

```
var miarray = new Array(3);
```



La propiedad length de las matrices nos indica el tamaño de éstas, es decir, su número de elementos.

## 5.c. Definición de matrices (continuación).-



Consiste en designar los elementos dentro de la propia creación de la matriz, separándolos con comas dentro de los paréntesis:

```
var miarray = new Array("elemento 1", "elemento 2", "elemento 3");
```



Hay que tener cuidado con este tipo de declaraciones ya que si solo queremos introducir un elemento y este es un número, el constructor de la matriz creerá que se trata de su dimensión (sintaxis anterior) y no de un elemento.

## 5.d. Acceso a los elementos .-

 Para acceder al contenido de un elemento de una matriz sólo debemos escribir el nombre de ésta seguido por la posición entre corchetes:

```
miarray[0] = "elemento 1"  
miarray[1] = "elemento 2"  
miarray[2] = "elemento3"
```

 Es importante recordar que en JavaScript las matrices siempre se empiezan a numerar en el cero. Por eso cuando recorramos la matriz en un bucle determinado debemos hacerlo desde 0 hasta  $n-1$  siendo  $n$  el tamaño de la matriz.

## 5.e. Matrices multidimensionales .-



JavaScript no soporta de manera directa las matrices con varias dimensiones. Ello no quiere decir que no las podamos usar, solo que las tenemos que definir con código:



Ejemplo para definir una matriz bidimensional de 5 elementos por dimensión:

```
var miarray2d = new Array(5);
for (i=0; i<=4; i++) {
    miarray2d [i] = new Array(5);
}
```



Si en lugar de dos dimensiones deseásemos utilizar más, solo tendríamos que repetir el proceso anterior por cada elemento de las sucesivas matrices que fuésemos añadiendo.

## 5.f. Acceso a matrices multidimensionales .-



Una vez definida una matriz multidimensional, para acceder a sus elementos sólo tenemos que escribir los índices correspondientes entre corchetes, sólo que en este caso tendremos que usar tantos índices como dimensiones tenga la matriz. Donde el primer número se refiere siempre a la fila, y el segundo a la columna.

```
miarray2d [0] [0] = "Primer elemento";
```

## 5.g. Métodos del objeto Array .-

- Al igual que con el objeto Math accedíamos a todas las funciones matemáticas que nos proporcionaba JavaScript.
- Con el objeto Array podemos acceder a todas las funciones estándar para poder trabajar con matrices.

## 5.g. El método join .-

- Permite obtener en una cadena de texto todos los elementos de una matriz, separándolos por el indicador que deseemos.
- Toma como parámetro opcional una cadena que actuará como separador de los elementos. Si no se especifica ningún carácter, se utiliza una cadena vacía, por lo que todos los elementos se devolverán pegados. Como ejemplo:

```
var miarray = new Array(0,1,2,3,4,5);
alert (miarray.join(" - "));
```

## 5.g. El método reverse .-



Cambia el orden de los elementos de la matriz. Lo que hace es devolver la misma matriz, pero con los elementos cambiados de orden, es decir, el primero de ellos al final y el último al principio.

```
var miarray = new Array(0,1,2,3,4,5);
alert (miarray + "\n" + miarray.reverse() +
      "\n" + miarray);
```

## 5.g. El método sort .-



Se encarga de ordenar los elementos de una matriz, descargándonos de ese trabajo a nosotros.



En su sintaxis más básica, este método no toma ningún parámetro y ordena los elementos de la matriz de forma ascendente:

```
var miarray = new Array(0,1,2,3,4,5);  
alert (miarray.sort());
```



Para ordenar descendenteamente números y datos estándar basta con usar los métodos sort y reverse consecutivamente.

```
miarray.sort();  
miarray.reverse();
```

## 5.g. El método concat .-

 Concatena el contenido de dos matrices para obtener una nueva que es el conjunto de los elementos de las anteriores.

 Ejemplo:

```
var a, b, c;  
a = new Array(0,1,2,3,4,5);  
b = new Array(6,7,8,9);  
c = a.concat(b);  
alert (c);
```

## 5.g. El método slice .-

- Permite obtener un subconjunto de los elementos de una matriz especificando el primero y el último de ellos que formarán parte de la nueva matriz.

```
var a = new Array (0, 1, 2, 3, 4, 5, 6, 7, 8, 9);  
var b = a.slice(5, 8);  
alert (b);
```

- Este método devuelve los elementos especificados incluyendo el primero de ellos, pero sin incluir el último. Así, en el ejemplo anterior obtendremos los números 5, 6 y 7, pero no el 8. De hecho, este segundo parámetro es opcional, y si no se proporciona slice devuelve todos los elementos de la matriz a partir del especificado e incluyendo este.

# 6. Los objetos estándar del navegador

- a.**Descripción
- b.**Objetos predefinidos
- C.**El árbol DOM del documento

# 6.a. Los objetos estándar del navegador



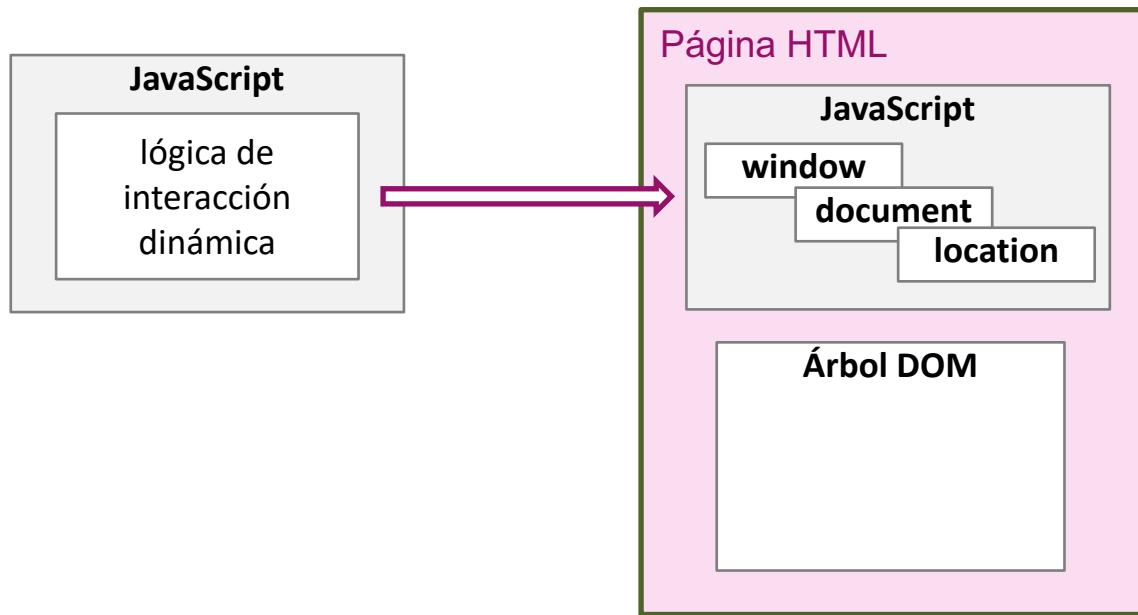
## Descripción



Hasta el momento, se ha descrito el lenguaje JavaScript, con su sintaxis básica y sus variables, con un enfoque de lenguaje de programación.



En este punto se describen los **elementos JavaScript** que están asociados a la página HTML, que son los utilizados para lograr las características dinámicas interactivas



## 6.b. Los objetos estándar del navegador



### Objetos predefinidos



Una página HTML tiene objetos JavaScript predefinidos, que permiten interactuar entre el contenido y el lenguaje:



**window**: referencia a la ventana propiamente tal. Todos los elementos pertenecen implícitamente al objeto window. Si se abre una nueva ventana o pestaña, corresponde a otro window.



**document**: es la raíz del documento HTML, y el "dueño" de todos los elementos. Provee atributos y métodos para acceder a los elementos de la página, lo que permite modificarlos dinámicamente, y de este modo realizar los efectos interactivos.

## 6.b. Los objetos estándar del navegador



### Objetos predefinidos

- **navigator**: contiene información sobre el navegador. Permite particularizar la programación, lo que es útil cuando existen diferencias entre los navegadores. También permite restringir una aplicación a un tipo o versión de navegador.
- **location**: contiene información sobre la URL de la página. Si se modifica, se abre una nueva URL, lo que permite utilizarlo para navegar programáticamente.
- **screen**: contiene información de la pantalla, incluyendo alto y ancho. Permite ajustar una página en función de la resolución de pantalla.
- **history**: contiene el historial de URLs visitadas, lo que permite volver atrás programáticamente.

## 6.c. Los objetos estándar del navegador

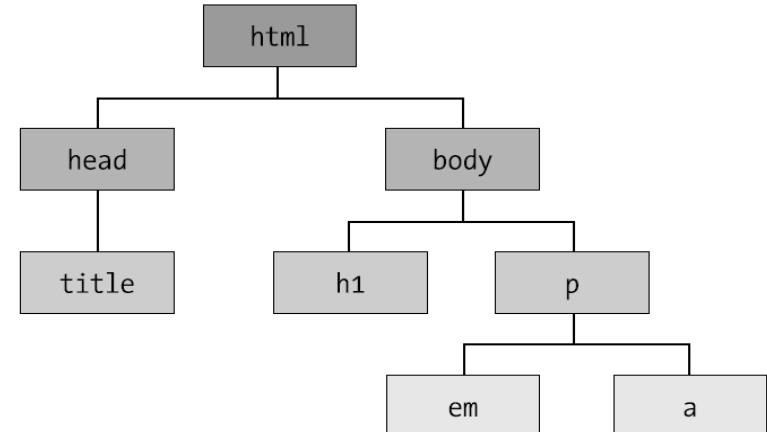


### El árbol DOM del documento



Es una ayuda ver la estructura de un documento HTML en forma de un árbol. El árbol comienza con el elemento raíz en la parte superior y todos los demás elementos parten hacia abajo desde esta raíz.

```
<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <h1>DOM</h1>
    <p>
      Ejemplo <em>árbol
      DOM</em>
      <a href="#">Volver</a>
    </p>
  </body>
</html>
```



# Ejercicio

-  Resumen del ejercicio:
-  Crear una nueva página HTML.
-  Crear una función JavaScript llamada onload, que se ejecuta cuando finaliza la carga de la página.
-  Obtener el nombre del navegador utilizado, y desplegarlo en un alert.
-  Obtener la resolución de pantalla del equipo, y desplegarlo en un alert.

# 7. Interacción entre páginas: El sistema de eventos

- a. Eventos
- b. Manejadores de eventos
- C. Eventos más comunes

## 7.a. Eventos .-

-  Los eventos son sucesos en nuestro documento que el navegador es capaz de captar, y responder en consecuencia, si hemos implementado la respuesta con código JavaScript.
-  Por ejemplo: un evento puede ser que el usuario, haga click sobre un botón (on Click), y en respuesta, nosotros podemos especificar un mensaje de aviso (alert):

```
<input type="button" value="Púlsar" onClick="alert ('¡Hola!');">
```

## 7.b. Manejadores de eventos .-



A las rutinas que un programador escribe para responder a un evento se les llama Manejadores de Eventos. También podemos decir que un manejador de eventos es una función que se lama de manera automática cuando ocurre una determinada acción en una página.



El nombre de estas funciones manejadores está formado por el nombre del evento, que pretenden controlar precedido por la palabra on. Así, el manejador para el evento Clic será onClick, para el evento Change será onChange y así sucesivamente.



Para que un elemento pueda responder a un evento se debe asignar al evento el manejador que se va a utilizar. El manejador puede ser una sucesión más o menos larga de instrucciones JavaScript, o directamente una función que hayamos definido para tal efecto.

## 7.b. Manejadores de eventos (continuación).-



La manera más habitual de asignar código para un evento consiste en introducir el nombre del evento como un atributo del objeto, asignándole como valor el nombre de la función manejadora.

```
<input type="button" value="Púlsar" onClick="alert ('¡Hola!');">
```



Pero, en cuanto nos excedamos de dos o tres instrucciones, no conviene escribirlas directamente en el manejador por razones de claridad y orden.



Es más adecuado escribir una función que se encargue de todo el proceso de invocarla desde el manejador.

```
<input type="button" value="Púlsar" onClick="boton_onClick();">
```

Objeto	Eventos
window	onLoad onUnLoad
	onBlur (perdida del foco)
	onFocus
link	onClick onMouseOut
	onMouseOver
area	onMouseOver onMouseOut
image	onAbort onError
	onLoad
form	onReset onSubmit
Text, textarea, password	onBlur onChange
	onFocus
	onSelect

## 7.c. Eventos más comunes .-

Objeto	Eventos
Button, reset, submit, radio, checkbox	onClick
select	onBlur onChange
	onFocus
Controles de selección de archivos (File)	onBlur onFocus
	onSelect

# 8. Gestión de formularios HTML



Cada campo de un formulario se puede manejar como un objeto JavaScript, permitiendo aplicar HTML dinámico. Por ejemplo, se tiene el siguiente formulario:

```
<form class="frm" >
  <div>
    <label for="summary">Summary:</label><input type="text" size="30"
      id="idSumm" name="summary" value="this is a summary">
  </div>
  <div>
    <label for="secret">Secret:</label><input type="password"
      name="secret" size="15" value="mypassword">
  </div>
  <div>
    <label for="type">Type:</label><select id="idType" name="type">
      <option value="1" selected="selected">
        Technology</option>
      <option value="2">Production</option>
    </select>
  </div>
</form>
```

# 8. Gestión de formularios HTML



Asignación de un valor a un campo de texto:

```
document.getElementById("idSumm").value = "other value";
```



Selección de la primera opción de un campo tipo select:

```
document.getElementById("idType").selectedIndex = 0;
```



Texto de la opción seleccionada de un campo tipo select:

```
document.getElementById("idType").options[  
    document.getElementById("idType").selectedIndex].text;
```

# 8. Gestión de formularios HTML



Agregar una nueva opción a un campo de tipo select:

```
var opt = new Option("Methodology", "3");
document.getElementById("idType").options[2] = opt;
```

options es un array con las opciones. Al agregarle un valor, se agrega la opción al select.

Se coloca el largo del array options.



Eliminar una opción de un campo de tipo select:

```
document.getElementById("idType").options[2] = null;
```

Al colocar null, se elimina la opción del array.

# 8. Gestión de formularios HTML



Dejar un campo como sólo lectura:

```
document.getElementById("idSumm").readOnly = true;
```



Deshabilitar un campo:

```
document.getElementById("idType").disabled = true;
```

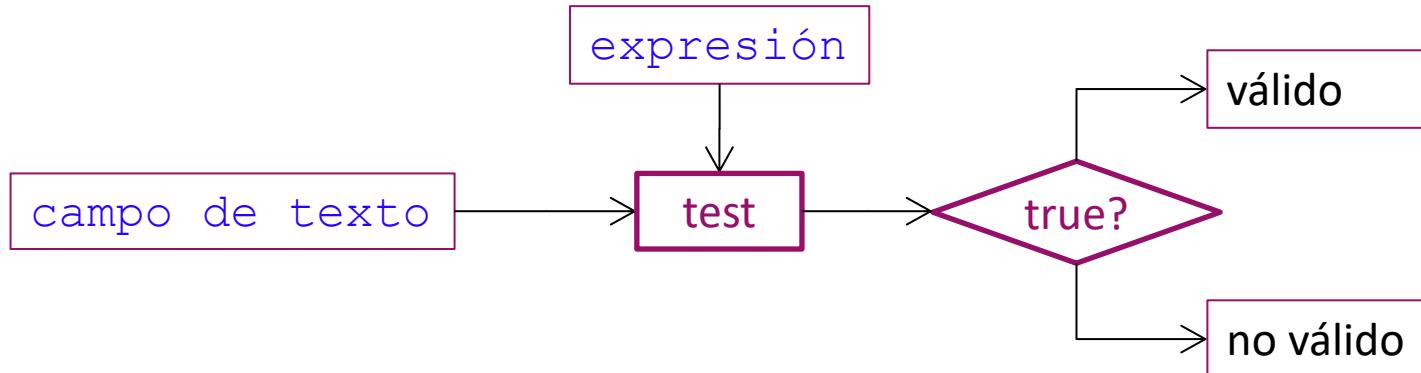
# 8. Gestión de formularios HTML



## Expresiones regulares



JavaScript permite utilizar expresiones regulares para la validación de textos, lo que resulta útil en el uso de formularios.



Implementación:

```
if (expr.test(field)) {  
    //válido  
} else {  
    //no válido  
}
```

# 8. Gestión de formularios HTML



## Expresiones regulares



Una expresión, aunque es una cadena, tiene una forma de codificarse que no necesita comillas.

- Comienzo: `/^`
- Final: `$/`



Para especificar tipos de caracteres:

- `\d`: dígito
- `\s`: espacio en blanco, tabulación o salto de línea
- `\w`: letra, número o `"_"`



Para especificar una cantidad de caracteres, se utiliza un número entre `{ y }`.

Por ejemplo, para validar un código postal de exactamente cinco dígitos, se utiliza:

```
var exp = /^{\d{5}}$/;
```

# 8. Gestión de formularios HTML



## Expresiones regulares



Si se quieren especificar varios grupos de caracteres, se colocan entre [ y ]:

- **[abc]**: busca las ocurrencias de "a", "b" o "c".
- **[a-e]**: busca las ocurrencias desde "a" a "e", es decir, "a", "b", "c", "d" o "e".
- **[a-eA-E]**: busca desde "a" a "e", o desde "A" a "E". **No** busca "eA".



Por ejemplo, para validar un nombre que tiene sólo letras, sin espacios, con largo mínimo 2 y máximo 20:

```
var exp = /^[a-zA-Z]{2,20}$/;
```



Si se quiere incluir la 'ñ' (que no está en el rango a-z), es:

```
var exp = /^[a-zA-ZñÑ]{2,20}$/;
```

# 8. Gestión de formularios HTML

## Expresiones regulares

 Si se quieren encontrar varias ocurrencias de una expresión, se pueden utilizar los cuantificadores:

- **+**: uno o más.
- **\***: cero o más.
- **?**: cero o una.

 Por ejemplo, para validar un campo con un número decimal positivo separado por coma, sin separador de miles, que admite '16,' y no admite ',15', es:

```
var exp = /^\\d+\\,(\\d*)?$/;
```

Uno o más dígitos

Una coma seguida de 0 o más dígitos

La coma y los decimales, 0 ó 1 vez.

# 8. Gestión de formularios HTML



## Expresiones regulares

Algunas otros comportamientos:

- La expresión "." (punto) equivale a cualquier carácter.
- Si se quiere buscar caracteres utilizados en las expresiones, como '\ ' o '. ', hay que escaparlos con un '\ '.
- Distintos valores de texto se pueden separar por '|'.



Por ejemplo, para validar una URL de un recurso, que puede tener como protocolo "http" o "file", seguida de una dirección separada por '\_' y '!', es:

```
var exp = /^(http|file):\/\/\w+(\.\w+)*$/;
```



Comienza con 'http'  
o 'file' seguido de ':'

Continúa con '//' . Cada  
'/' se escapa con un '\.'

Cada '.' se escapa  
con un '\.'

# 8. Gestión de formularios HTML



## Validación de formularios: Ejercicio



Dado un formulario con campos de distinto tipo:

- Completar los largos máximos
- Realizar validaciones cruzadas
- En un campo de tipo texto, validar valor con expresión regular

# Ejercicio 1.-



Realizar un formulario donde se pida el nombre y los apellidos al usuario.



Incluir un botón Aceptar que al pulsarlo llame a una función que muestre el siguiente mensaje :

“Hola ..... Bienvenido a mi página”

## Ejercicio 2.-



Repetir el ejercicio anterior pero esta vez que el mensaje de bienvenida se vea en un área de texto dentro del formulario al pasar el ratón sobre el botón de envío (evento onMouseOver).

# 9. Biblioteca estándar de JavaScript

 Aunque con HTML dinámico se pueden hacer muchos efectos interesantes, el esfuerzo asociado puede ser grande en algunos casos. En la práctica, es conveniente utilizar librerías y frameworks que ya los tienen implementados.

 Librerías:

- jQuery

 Frameworks de componentes:

- dhtmlxSuite
- Dojo toolkit
- jQuery UI

 Librerías para single-page application:

- backbone.js
- angular

# 10. Introducción a AJAX

- a.Que es Ajax
- b.Tecnologías Ajax
- C.El objeto XMLHttpRequest
- d.Modelo síncrono
- e.Modelo Asíncrono
- f.Obtener el objeto XMLHttpRequest
- g.Método open
- h.Método setRequestHeader
- i.Propiedad onreadystatechange
- j.Valores de readyState
- k.Método send

# 10.a. Introducción a AJAX



## Que es ajax



El termino AJAX, Asynchronous JavaScript and XML, hace referencia a una combinación de tecnologías y estándares que consiste en la solicitud asíncrona de datos al servidor desde una página web y la utilización de estos para actualizar una parte de la misma, sin obligar al navegador a realizar una recarga completa de toda la página.

# 10.b. Introducción a AJAX



## Tecnologías Ajax



Las tecnologías que nos permiten trabajar con AJAX son:

- **XHTML y CSS;** son los dos estándares definidos por el W3C para la construcción de páginas web.
- **JavaScript;** permite realizar solicitudes al servidor desde la página Web, recuperar los datos enviados en la respuesta y modificar el aspecto de la interfaz.
- **XML;** para el envío de datos de forma estructurada.
- **DOM;** permite organiza los elementos de la página de forma jerárquica.
- **El objeto XMLHttpRequest;** Es el componente esencial en AJAX.

## 10.c. Introducción a AJAX

-  **El objeto XMLHttpRequest**
-  Se trata del componente fundamental de una aplicación AJAX. A través de sus propiedades y métodos es posible lanzar peticiones en modo asíncrono al servidor y acceder a la cadena de texto enviada en la respuesta.
-  Para poderlo utilizar, deberá ser previamente instanciado desde la aplicación.

## 10.d. Introducción a AJAX



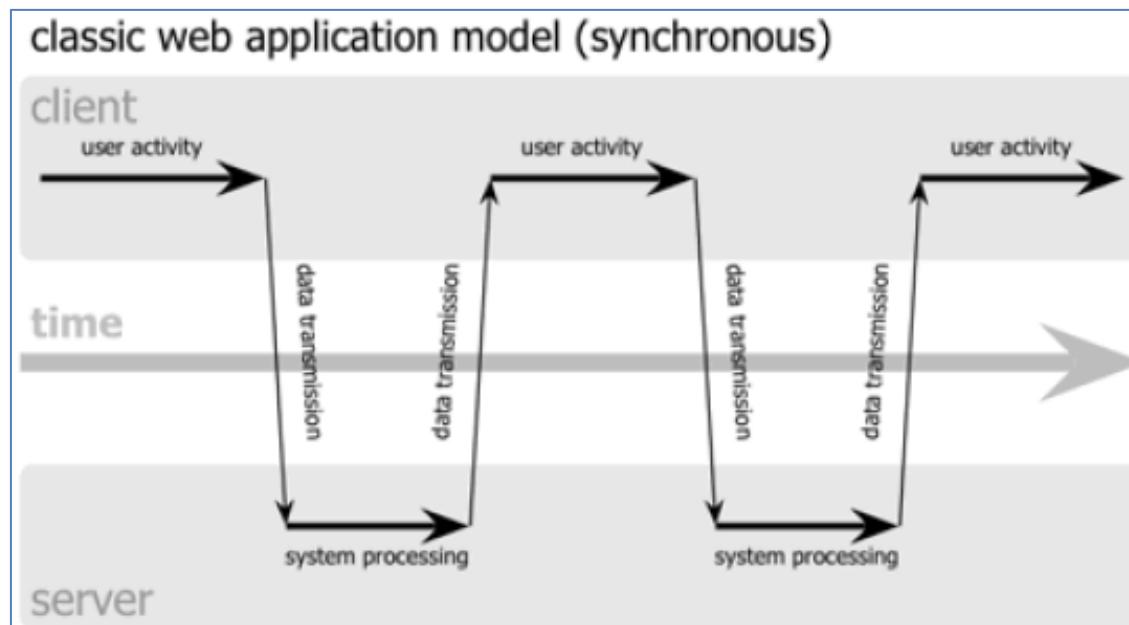
### Modelo síncrono



Como se puede apreciar en el siguiente diagrama, en el modelo síncrono o bien interactúa el usuario con la página o se llevan a cabo procesos en el lado del servidor.



Cuando interactúa el usuario o cliente, el servidor permanece inactivo. Sin embargo cuando se ejecutan los procesos en el servidor, la actividad del lado del cliente se detiene esperando recibir la respuesta a su petición.



# 10.e. Introducción a AJAX

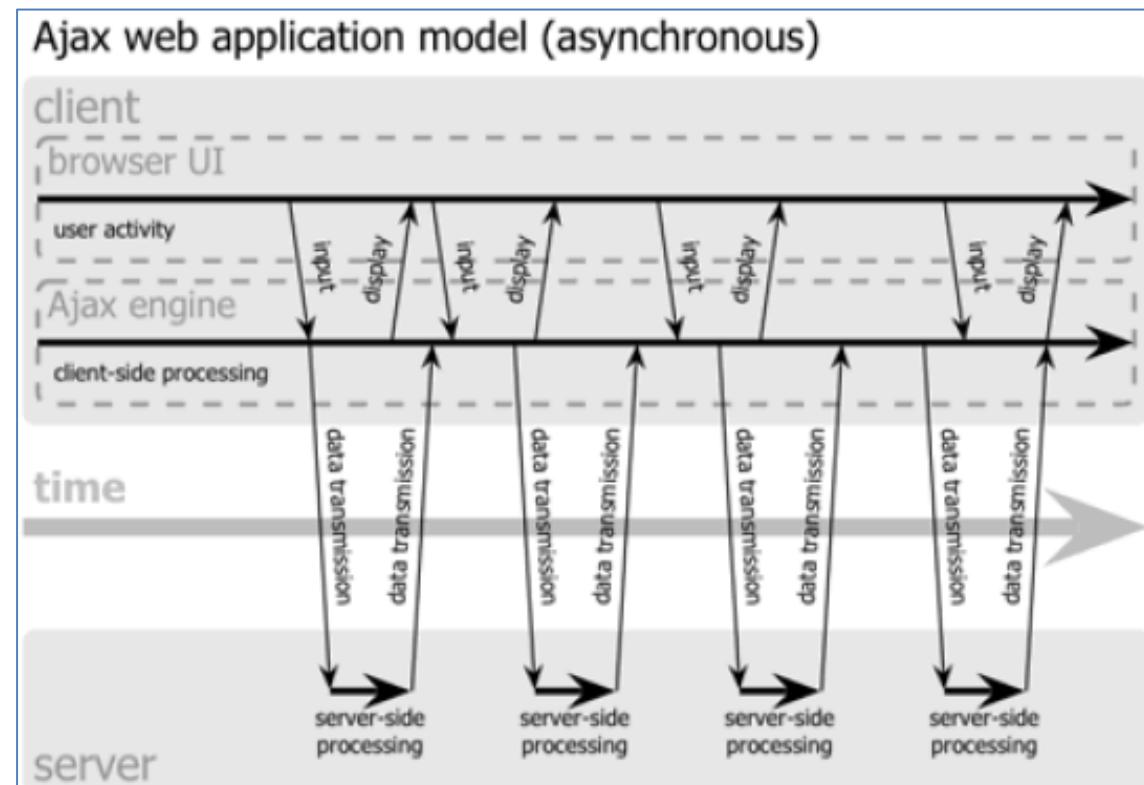


## Modelo asíncrono

Con el modelo asíncrono no existen períodos de inactividad del lado del cliente. Si la página emite una petición, esta será enviada al servidor para su procesamiento pero el cliente puede seguir interactuando en la página.



Una vez que el servidor envíe la respuesta, se actualizará parte de la página y el usuario verá reflejado el cambio.



## 10.f. Introducción a AJAX



### Obtener el objeto XMLHttpRequest



La forma de obtener este objeto es diferente dependiendo del navegador que se utilice:



ActiveXObject solo es reconocido por el navegador Internet Explorer



XMLHttpRequest es implementado como objeto nativo por el resto de navegadores, Opera, Firefox, Safari e incluso Internet Explorer a partir de la versión 7.



A continuación mostramos un fragmento donde se obtiene dicho objeto de una forma estándar para cualquier navegador.

# 10. Introducción a AJAX

```
function buscarSinopsis(){
    // solo para Explorer 6 y anteriores
    if(window.ActiveXObject)
        xhr = new ActiveXObject("Microsoft.XMLHttp");
    // resto de navegadores
    else if((window.XMLHttpRequest) ||
            (typeof XMLHttpRequest) != undefined)
        xhr = new XMLHttpRequest();
    // si el navegador no soporta AJAX
    else{
        alert("Su navegador no soporta AJAX");
        return;
    }
}
```

# 10.g. Introducción a AJAX



## Método open



Este método es el encargado de preparar la petición que enviaremos al servidor.

Es un método sobrecargado por lo cual existen varias versiones de él. La versión completa incluye 5 argumentos:

```
void open(in DOMString method,  
          in DOMString url,  
          in boolean async,  
          in DOMString user,  
          in DOMString password)
```

- ↳ method; indicamos el método de la petición. Lo más común es GET o POST
- ↳ url; La url donde enviamos la petición. Si utilizamos el método GET podemos añadir la lista de parámetros.
- ↳ async; indica si la petición es asíncrona (true) o síncrona (false)
- ↳ user; usuario valido para poder acceder a esa url
- ↳ password; la contraseña que permite acceso.



## 10.g. Introducción a AJAX



### Método open



En el siguiente código estamos utilizando el método GET para enviar la petición al servlet con url SinopsisLibros y además enviamos el parámetro indice.



El valor true indica que la petición será asíncrona.

```
// preparar la peticion  
// true --> peticion asincrona, false --> peticion sincrona  
xhr.open("GET", "SinopsisLibros?indice="+indice, true);
```

# 10.h. Introducción a AJAX



## Método setRequestHeader



Este método permite establecer cabeceras en la petición.

```
void setRequestHeader(in DOMString header,  
                      in DOMString value)
```



Ejemplo:

```
xhr.setRequestHeader("ACCEPT","text/xml");
```

# 10.i. Introducción a AJAX



**Propiedad onreadystatechange**



Esta propiedad la utilizamos para indicar qué función se encarga de procesar la respuesta.

Atribute EventListener onreadystatechange



Al estar definida como EventListener significa que debe contener la definición de una función manejadora de evento. El evento será el cambio de estado y se produce el evento varias veces desde que se envía la petición hasta la recepción de la respuesta.

```
// informar de la funcion que procesara la respuesta  
xhr.onreadystatechange = procesarRespuesta; // sin parentesis
```

# 10.j. Introducción a AJAX



## Valores de readyState

La función que procesa la respuesta será invocada cada vez que se produce un cambio de estado.

Desde que se crea el objeto XMLHttpRequest hasta que se recibe la respuesta por completo se producen 4 cambios de estado.

Para poder controlar que la función se procesa una sola vez, cuando se ha recibido por completo la respuesta la variable readyState nos ofrece los siguientes valores:

- 0 El objeto XMLHttpRequest se ha creado pero aún no se ha configurado la petición.
- 1 La petición se ha configurado pero aún no se ha enviado.
- 2 La petición se acaba de enviar, aunque aún no se ha recibido respuesta.
- 3 Se ha recibido la cabecera de la respuesta pero no el cuerpo
- 4 Se ha recibido el cuerpo de la respuesta. Es el momento en que esta puede procesarse.

```
function procesarRespuesta() {
    // si se ha recibido por completo la respuesta
    if(xhr.readyState == 4){
        document.getElementById("sinopsis").innerHTML =
            "<h4 bgcolor='red'>" + xhr.responseText + "</h4>";
    }
}
```

# 10.k. Introducción a AJAX



## Método send



Este método es el encargado de enviar la petición.

`void send()`



También es un método sobrecargado:

- `void send(in DOMString data);` se envía la lista de parametros en el cuerpo de la petición. Se utiliza para el método POST
- `void send(in Document data);` envia un documento XML en el cuerpo de la petición.

```
// enviar la peticion  
xhr.send(null);
```



En este ejemplo ponemos como argumento el valor null puesto que estamos utilizando el método GET y los parámetros los hemos concatenado a la url.

# Programación asíncrona

# Síncrono vs Asíncrono



Se refiere a cuando tendrá lugar la respuesta:

-  **Síncrono:** es frecuente emplear 'bloqueante' y 'síncrono' como sinónimos, dando a entender que toda la operación de entrada/salida se ejecuta de forma secuencial y, por tanto, debemos esperar a que se complete para procesar el resultado.
  
-  **Asíncrono:** la finalización de la operación *I/O* se señala más tarde, mediante un mecanismo específico como por ejemplo un *callback*, lo que hace posible que la respuesta sea procesada en diferido. Como se puede adivinar, su comportamiento es no bloqueante ya que la llamada *I/O* devuelve inmediatamente.

# Callbacks

-  Los *callbacks* son la pieza clave para que Javascript pueda funcionar de forma asíncrona. De hecho, el resto de patrones asíncronos en Javascript está basado en *callbacks* de un modo u otro.
-  Un *callback* no es más que **una función que se pasa como argumento de otra función**, y que será invocada para completar algún tipo de acción.
-  En nuestro contexto asíncrono, un *callback* representa el '*¿Qué quieres hacer una vez que tu operación asíncrona termine?*'. Por tanto, es el trozo de código que será ejecutado una vez que una operación asíncrona notifique que ha terminado.

```
var miCallback = () => {alert("Adios")};  
setTimeout(miCallback, 1000);
```

# Promesas

-  Una promesa es un objeto que representa **el resultado de una operación asíncrona**. Este resultado podría estar disponible **ahora** o en el **futuro**. Las promesas se basan en *callbacks*.
-  Cuando llamamos a una función asíncrona implementada con este patrón, nos devolverá inmediatamente una promesa como garantía de que la operación asíncrona finalizará en algún momento, ya sea con éxito o con fallo.
-  Una vez que tengamos el objeto promesa en nuestro poder, registramos un par de *callbacks*: uno para indicarle a la promesa '*que debe hacer en caso de que todo vaya bien*' (resolución de la promesa o *resolve*) y otro para determinar '*que hacer en caso de fallo*' (rechazo de la promesa o *reject*).

```
var url = "https://pokeapi.co/api/v2/pokemon/bulbasaur";  
var promesa = fetch(url);  
promesa.then(async (resultado) => console.log(await resultado.json()));  
promesa.catch(error => console.log(error));
```

# Async / Await

-  Las palabras clave **async** y **await** surgieron para simplificar el manejo de las promesas. Son para hacer las promesas más amigables, escribir código más sencillo, reducir el anidamiento y mejorar la trazabilidad al depurar. Pero recuerda, **async \ await** y las promesas son lo mismo en el fondo.
-  La etiqueta **async** declara una función como asíncrona e indica que una promesa será automáticamente devuelta. Podemos declarar como **async** tanto funciones con nombre, anónimas, o funciones flecha.
-  Por otro lado, **await** debe ser usado siempre dentro de una función declarada como **async** y esperará automáticamente (de forma asíncrona y no bloqueante) a que una promesa se resuelva.

# Async / Await

```
fetch("https://pokeapi.co/api/v2/pokemon/bulbasaur")
  .then(async (response) => {
    const data = await response.json();
    console.log(data);
  })
  .catch(error => console.log(error));
```

# DOCUMENTACIÓN

## DOCUMENTACIÓN

- [Características javascript](#)
- [Guia oficial](#)
- [Documentacion mozilla](#)
- [Consulta API](#)

## DOCUMENTACIÓN ADICIONAL

- [Tutorial](#)
- [Libros gratuitos de JavaScript](#)

## TEST NAVEGADORES HTML 5

- [Test Navegadores](#)

## HERRAMIENTAS UTILIZADAS EN EL CURSO

- [Visual Studio Code](#)

**Todos  
aprendemos.  
Gracias!!**



**¡Seguimos en contacto!**

---

[www.iconotc.com](http://www.iconotc.com)

