Actividad de Aprendizaje N° 04

Desarrollo FrontEnd con ECMAScript

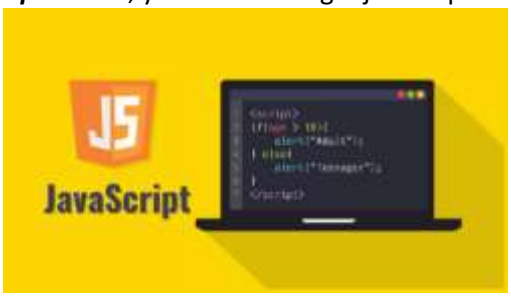
Compiled by: Mg. Jaime Suasnábar Terrel

1. JavaScript

JavaScript es un lenguaje de programación script Técnicamente, *JavaScript es un lenguaje script dinámico, multi-paradigma*, basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa.

JavaScript se ha convertido en el ***leguaje mas importante***, ya no es un lenguaje solo para la web sino JavaScript nos permite crear:

- Aplicaciones de servidor
- Aplicaciones móviles
- Aplicaciones de escritorio
- Aplicaciones de consola
- Aplicaciones de robótica
- Aplicaciones empotradas(smartTV)
- Aplicaciones web



Si bien en un principio este lenguaje se usó para crear animaciones en la web, con el lanzamiento de **nodeJS**, la industria cambió y las posibilidades crecieron, prácticamente, sin conocer límites.

En una encuesta realizada por la *HackerRank* a 71.000 desarrolladores de 100 países, el 71% dominan Java, y el *73% Javascript*, por lo que JAVASCRIPT es el lenguaje de programación más popular. ***Curiosamente, solo el 42% de los estudiantes de informática lo dominan***, indicando que es un lenguaje que se aprende en el trabajo, no en la Universidad.

1.1. CONOCIENDO JAVA SCRIPT

El estándar de JavaScript es ECMAScript. Desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1. Los navegadores más antiguos soportan por lo menos ECMAScript 3. El 17 de Julio de 2015, ECMA International publicó la sexta versión de ECMAScript, la cual es oficialmente llamada ECMAScript 2015, y fue inicialmente nombrada como ECMAScript 6 o ES6. Desde entonces, los estándares ECMAScript están en ciclos de lanzamiento anuales. Esta documentación se refiere a la última versión del borrador, que actualmente es ECMAScript 2019.

Las versiones de JavaScript en los diversos navegadores no es la misma, cada navegador cuenta con su propia implementación.

Implementación Mozilla.org

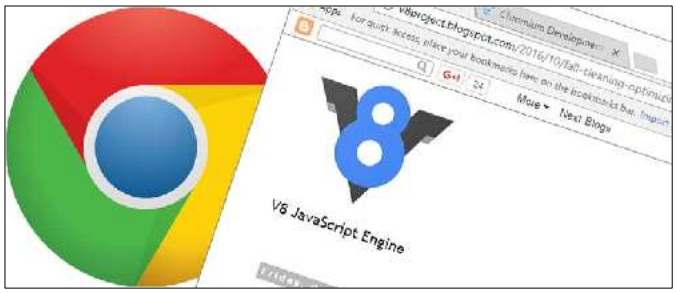
La primera implementación de JavaScript fue creada por Brendan Eich en Netscape, y desde entonces ha sido actualizada (en JavaScript 1.5) para cumplir con ECMA-262 Edición 5. Este motor, cuyo nombre en código es ***SpiderMonkey***, se implementa en C. El motor Rhino, creado principalmente por Norris Boyd (también en Netscape) es una implementación de Javascript en Java. Al igual que SpiderMonkey, Rhino cumple con ECMA-262 Edición 3.

La segunda implementación se denomina ***TraceMonkey*** (Firefox 3.5), JägerMonkey (Firefox 4) e IonMonkey fueron agregadas al motor de JavaScript de SpiderMonkey durante el tiempo.

Otras Implementación

Existen otros motores de JavaScript populares tales como:

- **V8 de Google**, el cual es usado en el navegador Google Chrome.



- **JavaScriptCore** (SquirrelFish/Nitro) usado en algunos navegadores basados en WebKit tales como Apple Safari.
- **Carakan** en Opera.

z

1.2. ¿QUÉ ES NODE JS?

Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. https://nodejs.org/es/

Fue creado por **Ryan Dahl** en 2009 y su evolución está apadrinada por la empresa Joyent, que además tiene contratado a Dahl en plantilla.

Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor. Node.js implementa algunas especificaciones de CommonJS.7 Node.js incluye un entorno REPL para depuración interactiva.

2. Conceptos básicos

JavaScript está influenciado por la sintaxis de Java, Awk, Perl y Python.

En JavaScript, las ***instrucciones*** son llamadas ***Sentencias*** y son **separadas por un punto y coma (;)**. Un punto y coma no es necesario al final de una sentencia si está escrita en una sólo línea. Sin embargo, es recomendable siempre terminar con punto y coma. JavaScript es case-sensitive y utiliza el conjunto de caracteres Unicode.

2.1. ¿Dónde INCLUIR javascript?

JavaScript se puede incluir en tres áreas bien diferenciadas.

Incluir javascript en el mismo documento html

```
<script>
  Codigo Java Script
</script>
```

Definir javascript en un archivo externo

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

Archivo: código.js

```
/*Colocar código java script/*
```

Incluir javascript en tag

Se incluye JavaScript en el HTML de la página:

```
<body
<p onclick="alert('Mensaje de prueba')">Párrafo Vinculo </p>
</body>
```

2.2. Comentarios en JavaScript y PHP

La sintaxis es: (Igual a Java)

```
// Comentario en una sola línea

/* Este es un comentario Multilínea
 */
```

3. VARIABLES Y OPERADORES

3.1. Variables. (DIFERENTE A JAVA Y PHP)

Son espacios de memoria con valores y tienen un nombre. El nombre de una variable tiene que empezar con ***una letra, un guión bajo (_) o un símbolo de dólar ($)***; los valores subsiguientes pueden ser números. Puedes usar la ISO 8859-1 o letras Unicode tales como å y ü.

Tipos de variables

Variable global o local (Depende donde es declarado)

Sintaxis:

```
var Nom_Variable;
```

o

```
var Una_Variable = "Texto";
```

Variable local

Sintaxis:

```
let Nom_Variable;
```

o

```
let Una_Variable = "Texto";
```

Constante

Usar const en:

- Una nueva matriz
- Un nuevo objeto
- Una nueva función
- Una nueva expresión regular

Sintaxis:

```
const Nom_constante;
```

o

```
const Nom_constante = "Texto";
```

NO puedes:	Pero puedes:
➢ Reasignar un valor constante ➢ Reasignar una matriz constante ➢ Reasignar un objeto constante	➢ Cambiar los elementos de la matriz constante ➢ Cambiar las propiedades del objeto constante

Puede usar **undefined** para comparar si una variable tiene un valor. En el siguiente código a la variable input no se le asigna ningún valor y la sentencia de control if la evalúa como true.

```
var input;
if(input === undefined){
  hazEsto();
} else {
  hazEso();
}
```

El valor **undefined** ***se comporta como un*** **false** cuando se utiliza en un contexto booleano. Por ejemplo, el siguiente código ejecuta la función myFunction porque el elemento myArray no ha sido definido:

```
var myArray = new Array();
if (!myArray[0]) myFunction();
```

El valor **undefined** se convierte en **NaN**, no numérico, cuando se usa en una operación aritmética.

```
var a;
a + 2; // Se evalua a NaN
```

Cuando se evalúa una variable nula, el valor null se comporta como el 0 en operaciones aritméticas y como false en operaciones lógicas. Por ejemplo:

```
var n = null;
console.log(n * 32); // Va a lanzar 0 a la consola
```

Variables Globales

Las variables globales son propiedades del objeto global. En las páginas web, **el objeto global es *window***, se puede establecer y acceder a las variables globales usando la sintaxis ***window.variable***.

Puedes acceder a variables globales declaradas en una ventana o frame de otra ventana o frame… *especificando el nombre de la ventana o frame.*

Ejemplo: Si una variable denominada phoneNumber es declarada en un documento, puede consultar a esta variable desde un iframe como ***parent.phoneNumber***.

3.2. Tipos de datos

El último estándar ECMAScript define ocho tipos de datos:

Siete tipos de datos que son primitivos:

- Boolean
- Null
- Undefined
- Number
- BigInt
- String
- Symbol
- Object

Boolean

Este tipo de dato almacena un bit que indica true o false.

```
var si = true; var no = false;
```

null

Valor nulo. null

```
var x = null;
```

undefined.

Una propiedad de alto nivel cuyo valor no es definido.

```
var dato; // su valor es undefined var dato = undefined;
```

Number

Un número entero o un número con coma flotante.

```
var miEntero = 1; var miDecimal = 1.33;
```

BigInt

Un número entero con precisión arbitraria.

```
var miEntero = 9007199254740992;
```

String

Las variables de tipo string almacenan caracteres o palabras.

```
var dato = "Esto es un string";
var otroDato = 'Esto es otro string';
```

Symbol

Permite obtener valores que no pueden volver a ser creados, es decir, son identificadores únicos e inmutables.

```
const myFirstSymbol = Symbol();
```

Object

Un objeto es una *estructura de datos compleja* en memoria identificada con un nombre. Es una colección de propiedades.

- Cuando se **declara un objeto**, *un limitado grupo de propiedades son inicializadas*; luego pueden ser agregadas o eliminadas otras propiedades.
- Los ***valores de las propiedades*** pueden ser de **cualquier tipo**, incluyendo otros **objetos** lo cual permite construir estructuras de datos más complejas.
- Las ***propiedades*** se identifican usando ***claves***. Una clave es un valor String o Symbol.

Tipos de propiedades de objetos:

Propiedad de datos

Asocia una clave con un valor

Atributo	Tipo	Descripción	Valor por defecto
[[Value]]	Cualquier tipo JavaScript	El valor obtenido mediante un acceso get a la propiedad.	undefined
[[Writable]]	Boolean	Si es false, el [[Value]] de la propiedad no puede ser cambiado.	false
[[Enumerable]]	Boolean	Si es true, la propiedad será enumerada en ciclos for...in. Consultar también Enumerabilidad y pertenencia de las propiedades	false
[[Configurable]]	Boolean	Si es false, la propiedad no puede ser eliminada y otros atributos que no sean [[Value]] y [[Writable]] no pueden ser cambiados.	false

Propiedad de acceso

Asocia una clave con una o más funciones de acceso (get y set) para obtener o almacenar un valor

Atributo	Tipo	Descripción	Valor por defecto
[[Get]]	Función, objeto o undefined	La función es llamada con una lista de argumentos vacía y devuelve el valor de la propiedad cada vez que un acceso get al valor es realizado. Consultar también get.	undefined
[[Set]]	Function object or undefined	La función es llamada con un argumento que contiene la variable asignada y es ejecutada siempre que una propiedad específica se intenta cambiar. Consultar también set.	undefined
[[Enumerable]]	Boolean	Si el valor es true, la propiedad será enumerada en bucles for...in.	false
[[Configurable]]	Boolean	Si el valor esfalse, la propiedad no podrá ser eliminada y tampoco ser modificada a una propiedad de datos.	false

En JavaScript todo es un objeto, hasta las funciones. Todo hereda de la clase Object heredan de Object tenemos Array , Date , etc...

3.3. Variable hoisting

Otra cosa inusual acerca de las variables en JavaScript es que pueden hacer referencia a una variable declarada más tarde, sin obtener una excepción. Este concepto se conoce como hoisting; Las variables en JavaScript son en cierto sentido "elevadas" a la parte superior de la función o declaración. Sin embargo, las variables que no se han inicializado todavía devolverán un valor undefined.

3.4. Operadores (IGUAL EN JAVA Y PHP)

Los operadores son símbolos que denotan operaciones que pueden realizarse entre elementos llamados operandos.

Aritmeticos						
+	-	*	/	%	++	--

Cadena		
+	Concatenacion.	Nota: ***"cadena "+15*** el resultado es ***"cadena 15".*** ***"37"-7 resultado 30*** ***"37"+7 resultado "377"***

Lógicos					
&&	Y	||	O	!	Negacion

Comparación			
>	>=	<	<=
==	**Igualdad** Devuelve true si ambos operadorandos son iguales.	===	**Estrictamente iguales** Devuelve true si los operandos son iguales y tienen el mismo tipo.
!=	Desigualdad	(!==)	Estrictamente desiguales

Asignación					
Operador	**Significado**	**Operador**	**Significado**	**Operador**	**Significado**
x = y	x = y	x *= y	x = x * y	x %= y	x = x % y
x += y	x = x + y	x /= y	x = x / y	x **= y	x = x ** y
x -= y	x = x - y				

Condicional (ternario)	
Condición ? valor1: valor2	Si la condición es true, devuelve el valor1, de lo contrario el valor2.
Ejemplo	`var estado = (edad >= 18) ? "adulto" : "menor";`

Operadores Unarios	

delete nombreObjeto;	Elimina un objeto, una propiedad de un objeto, o un elemento de un Array.
	`delete miObj.h;`
	`delete miObj;`

typeof operando
 	
typeof (operando)
	Devuelve una cadena con ***el tipo de dato*** del operando
	`typeof true; // "boolean"`
	`typeof "Texto"; //"string"`
	`typeof 354; //"number"`

void (expresion)
o
void expresion
	Especifica una expresión que será evaluada y no retornará ningún resultado.
	`<a href="javascript:void(0)"> No hace nada</a>`
	`<a href="javascript:void(document.form.submit())">`
	`Haz click para enviar</a>`
	`<a`
	`href="javascript:void(document.body.style.backgroundColor='red')">`
	`Color Fondo</a>`

3.5. Operadores Spred

Se representa con tres puntos: **...**

El operador spread permite expandir elementos iterables, como por ejemplo:

- arrays
- cadenas de texto
- objetos
- listas de nodos DOM (NodeList)
- el objeto arguments de una función
- y otros dos tipos de estructuras nuevas en ES6: maps y sets.

```
//Dado el siguiente array:

    const alfabeto = ["a","b","c","d","e"];

//Si lo sacamos expandido en consola:

    console.log(...alfabeto);//  a b c d e f
```

Ejercicio: Copiar un array

```
const array1 = ["A","B","C"]
const array2 = [...array1];
```

Ejercicio: Unir 2 arrays

```
const array1 = ["A","B","C"]
const array2 = ["D","E","F"]

const array3 = [...array1, ...array2];
```

Ejercicio: Expandir un String

```
//Si desplegamos en consola expandida dentro de un array
console.log([..."Texto"]);
```

```
//["T", "e", "x", "t", "o"]
```

1.1. EJERCICIO

A continuación vemos un ejemplo de operaciones.

```
// Ejercicio 01
// Realice un código para mostrar resultados de los operadores binarios
var a = 10, b = 6;

console.log('a = ' + a + ', b = ' + b + '<P>');
console.log('a + b = ' + (a + b));
console.log('a - b = ' + (a - b));
console.log('a * b = ' + (a * b));
console.log('a / b = ' + (a / b));
console.log('a % b = ' + (a % b));
console.log('a++ : ' + (a++));
/* Notar que aquí ya se ha incrementado 'a' */
console.log('-a = ' + (-a));

console.log(' a += b : a = ' + (a += b));
console.log(' a %= b : a = ' + (a %= b));
console.log(' a %= b : a = ' + (a %= b));
```

```
// Ejercicio 02
// Realice un código para mostrar excelente, notable y aprobado.

<!DOCTYPE html>
<html>
<head><meta charset="utf-8"/>
<script>
let notal, nota2, nota3;
let media;
datol = prompt("Primer numero?", "@");
dato2 = prompt("Segundo numero?", "@");
dato3 = prompt("Tercer numero?", "@");
notal = parseInt(dato1);
nota2 = parseInt(dato2);
nota3 = parseInt(dato3);
media = (notal + nota2 + nota3)/3;
if (media >16)
{ resultat = " Excelente";
}
else
{ if (media >=14)
resultat = " Notable";
else
resultat = " Aprobado";
}
document.write('La nota final es ${media} y eso es ${resultat}');
</script>
</body>
</html>
```

2. ESTRUCTURAS DE CONTROL (CONDICIONALES)

En este apartado veremos:

- El condicional if
- El condicional switch

Valores falsos:

Los siguientes valores se evalúan como falso (también conocidos como valores Falsy):

- false
- undefined
- null
- 0
- NaN
- la cadena vacía ("")

Valores verdaderos:

El resto de valores, incluídos todos los objetos, son evaluados como verdadero cuando son pasados a una sentencia condicional.

2.1. El condicional if (Igual en PHP)

La estructura del condicional sigue esta sintaxis en JavaScript:

```
if(condicion) {
     codigo necesario }
else {
     codigo alternativo }
```

2.2. El condicional switch (Igual en PHP)

También tenemos el condicional switch:

```
switch(opcion) {
  case caso1 :
       sentencias para caso1;
       break;
       ......
  case casoN :
       sentencias para casoN;
       break;
  default :
       sentencias por defecto;
       break;
  }
```

3. ESTRUCTURAS REPETITIVAS O BUCLES

En este apartado estudiamos las distintas construcciones que nos ofrece JavaScript para programar bucles:

- El bucle for
- El bucle while
- El bucle do ... while
- El bucle for...each
- El bucle for ... in

3.1. BUCLE FOR (Igual en PHP)

```
for([Inicialización]; [Condición]; [Expresión de actualización]){
     Instrucciones a repetir
   }
```

3.2. EL BUCLE WHILE (Igual en PHP)

```
while(Condición) {
   Instrucciones a repetir
}
```

3.3. EL BUCLE DO ... WHILE (Igual en PHP)

```
do {
   Instrucciones a repetir
} while(condicion);
```

```
var text = ""
var i = 0;
do {
    text += "<br>El número es: " + i;
    i++;
}while (i < 5);
document.getElementById("demo").innerHTML = text;
}
```

3.4. FOREACH (Este bucle se introdujo en ECMAScript5). (Diferente en PHP)

Es un método de la clase Array permite iterar un array de una manera secuencial.

Syntaxis:

```
Nom_array.forEach(nomFuncion, thisValue);
function nomFuncion(elemento, indice, arr){
    Codigo
};
```

Elemento	Requerido. El valor del actual valor
indice	Opcional. El array indice de los elementos
arr	Opcional el objeto array que pertenece al elemento actual

Ejemplo:

```
var miArray = [1, 2, 3, 4];
miArray.forEach(function (valor, indice) {
    console.log(indice + " : " + valor);
});
// Devuelve lo siguiente
// 0 : 1
// 1 : 2
// 2 : 3
```

Ejemplo:

```
var numeros = [65, 44, 12, 4];
numeros.forEach(miFuncion)

function miFuncion(item, index, arr) {
  arr[index] = item * 10;
}
```

```
console.log(números);
// [650, 440, 120, 40]
```

3.5. Bucle For/Of

Permite iterar sobre objetos iterables (Array, Map, Set, argumentos, objetos etc)

La sintaxis

```
for (variable of iterable) {
  Sentencias
}
```

Siendo:

Variable: En cada iteración el elemento (propiedad enumerable) correspondiente es asignado a variable.

Iterable: Objeto cuyas propiedades enumerables son iteradas.

Ejemplos

```
let iterable = [10, 20, 30];

for (let valor of iterable) {
    console.log(valor);
}
// 10
// 20
// 30
```

Iterando un String

```
let iterable = "fis";

for (let valor of iterable) {
  console.log(valor);
}
// "f"
// "i"
// "s"
```

Iterando un Map

```
let iterable = new Map([["a", 1], ["b", 2], ["c", 3]]);

for (let entrada of iterable) {
  console.log(entrada);
}
// ['a', 1]
// ['b', 2]
// ['c', 3]

for (let [indice, valor] of iterable) {
  console.log(valor);
}
// 1
// 2
// 3
```

Iterando un objeto arguments

```
(function() {
  for (let argument of arguments) {
    console.log(argument);
  }
})(1, 2, 3);
```

```
// 1
// 2
// 3
```

3.6. Bucle For/In

Permite iterar entre las propiedades de un objeto.

La sintaxis

```
for( indice in objeto )
```

Siendo:
Indice el nombre de la propiedad
objeto[indice] el valor de la propiedad.

Ejemplo práctico:

```
var cliente = {
     nombre: "Juan Perez",
     direccion: "Av. Honorio Delgado 920",
     telefono: "012705689",
     ciudad: "Lima"
};
for(var indice in cliente) {
     console.log(indice+" : "+cliente[indice] );
}
// Devuelve:
// nombre : Juan Perez
// dirección : Av. Honorio Delgado 920
// teléfono : 012705689
// ciudad : Lima
```

4. FUNCIONES Y PROCEDIMIENTOS

Una función de JavaScript es un bloque de código diseñado para realizar una tarea en particular y se ejecuta cuando lo invocan (o lo llaman).

- Los argumentos se pasan por valor
- Los objetos se pasan por referencia

4.1. Sintaxis de una Función

Se define con la palabra reservada ***function***, seguida de: su nombre, paréntesis () y bloque de código.

Sintaxis

```
function NombreFuncion(param1, ..., paramN){
    Código de la función

    return Valor;
}
```

NombreFunción pueden contener letras, dígitos, subrayados y signos de dólar (las mismas reglas que las variables).
Los paréntesis incluyen nombres de parámetros: param1, param2, …pramN separados por comas.

Bloque de código es el código que se ejecutará, se coloca dentro de llaves: {}

Las funciones a menudo calculan un valor de retorno ***return valor;*** El valor devuelto se "devuelve" al que lo llamó y la función dejará de ejecutarse

4.2. Invocación de funciones

Utilizamos el nombre de la función seguida del operador () dentro puede ir la lista de argumentos. Sino utilizamos es operador () devuelve la función como objeto. Una función se puede invocar en tres escenarios:

- Se ***invoca*** (llama) desde el código JavaScript
- Ocurre un ***evento*** (cuando un usuario hace clic en un botón)
- Automáticamente (***auto invocado***)

Llamar una función desde javascript

```
Var f1 = NombreFuncion(arg1,...);
```

Llamar una función desde un evento de html

Podremos llamar a una función (definida en <head> del HMTL) con cualquier tag, por ejemplo:

```
<p onClick="NombreFuncion(arg1,...);">Hacer Click Aqui</p>
```

4.3. Funciones Anónimas o una función como expresión

Son funciones que no tendrá un nombre, pero deben ser almacenadas en una variable.

Función normal

```
function nomFunccion(a, b) {   return a * b; }
```

Para llamar utilizo: nomFuncion(3,5);

¿Qué suscede si la función se asigna a una variable?

```
var x = function nomFunccion(a, b) {   return a * b; }
```

Para llamar a función utilizo: x(3,5) y ya no puedo usar nomFunción(3,5)

Función anónima

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

¿Cómo llamar a la función anónima?

Se *invocan (llaman) utilizando el nombre de la variable*. Es por ello que las funciones almacenadas en variables no necesitan nombres de funciones.

4.4. La Función como Objeto

Las funciones también se pueden definir con un constructor de funciones de JavaScript incorporado llamado Function().

Ejemplo

```
var myFunction = new Function("a", "b", "return a * b");
var x = myFunction(4, 3);
```

El typeofoperador en JavaScript devuelve "función" para funciones.

En realidad no tiene que usar el constructor de funciones. El ejemplo anterior es lo mismo que escribir:
Ejemplo

```
var myFunction = function (a, b) {return a * b};
var x = myFunction(4, 3);
```

4.5. Funciones Auto Invocadas

Es una función que se invoca (inicia) automáticamente, sin ser llamada. Para auto invocar una función utilizamos el operador ().
¿Porqué auto invocar a una función?
Rpta. Porque la función será usada una vez.

Sintaxis:

```
(Función)(Parámetros);
```

```
(function () {    //codigo   })(param1, param2, … paramN);
```

Se debe agregar paréntesis alrededor de la función para indicar que es una expresión de función:

```
Ejemplo
(function () {
  var x = "Hola!!";  // Función auto invocada
})();
```

4.6. Funciones de flecha

Son funciones que trata de simplifican el código. La sintaxis que utiliza es muy corta y generalmente serán utilizadas para funciones que contegan poco código.
Funcion anónima

```
const x = function(x, y) {   return x * y; }
```

Función flecha de varias sentencias

Es una función anónima sin la palabra reservada *function* pero contiene un arrow o flecha =>.

,
Sintaxis:

```
const x = (param1, …) => {   sentencias; }
```

Esta versión permite usar varas sentencias

Ejemplo:

```
const x = (x, y) => {
  let p = x * y;
  return p
};
```

Función flecha de una sola sentencia

Es una función anónima sin la palabra reservada *function, retun y las llaves* también contiene un arrow o flecha =>.

```
var x = (param1, …) => sentencia;
```

Si sentencia devuelve un valor la función flecha realiza un return de sentencia.

.

Ejemplo

```
// ES5
var x = function(x, y) {
  return x * y;
}
```

```
// ES6
const x = (x, y) => x * y;    // Hace return x * y
```

Ejemplo hace lo mismo que el anterior.

```
var x = (x, y) => return x * y;
```

4.7. El Objeto Arguments

Las funciones de JavaScript tienen un objeto incorporado llamado ***arguments*** que contiene la lista de argumentos de la función.

¿Cuál es la diferencia entre parámetro y argumento de una función?

- En una declaración de función los valores que reciben se denominan es parámetros de la función.
- En la invocación de función los valores que se envían se denominan argumentos de la función.

Argumentos de la Función

```
// Declaración
function NombreFuncion(param1, ..., paramN){
    Código de la función

    return Valor;
}

// Invocación
NombreFuncion(argumento1, argumento2, ..., argumentoM);
```

Por lo tanto Arguments tentrá una estructura de datos siguiente:

```
{ '0': argumento1, '1': argumento2, ...'m': argumentoM }
```

Ejemplo: Si invocamos la función *arguments* se muestra como un objeto.

```
// Declaramos una función
function unaFuncion(){
   console.log(arguments);
}
// Invocamos a la función
unaFuncion(1, 2, 3);

// Resultado
// [Arguments] { '0': 1, '1': 2, '2': 3}
```

Obtener Argumentos
Utilizamos el objeto arguments y sus indices
arguments[0]
arguments[1]
arguments[2]

Parámetros de la Función

Los parámetros son las variables locales listadas en la declaración de la función y pueden tomar un valor por defecto.

```
// Función con valor por defect ES6
function unaFuncion(a=valorDefecto1, b=valorDefecto2){ codigo; }
```

A cada parámetro se la asigna un argumento automáticamente perdiendo el valor por defecto.
Parametro1=argumento1
Parametro2=argumento2
Asi sucesivamente…

En JavaScript no es necesario que la cantidad de argumentos sea igual a la de parámetros.

Ejemplo: Función con más parámetros

```
// Declaramos una función
function unaFuncion(a, b, c){
   console.log(arguments);
}
// Invocamos a la función
unaFuncion(2, 3);

// Resultado
// [Arguments] { '0': 2, '1': 3 }
```

Parametro1=argumento1
Parametro2=argumento2
Parametro3=undefined

Ejemplo: Función con más argumentos

```
// Declaramos una función
function unaFuncion(a, b){
   console.log(arguments);
}
// Invocamos a la función
unaFuncion(2, 3, 4, 5);

// Resultado
// [Arguments] { '0': 2, '1': 3, '2': 4, '2': 5 }
```

Parametro1=argumento1
Parametro2=argumento2
Ignorado argumento3, argumento4, …

Ejemplo

```
x = sumarTodo(1, 123, 500, 115, 44, 88);
```

```
function sumarTodo() {
  var i;
  var sum = 0;
  for (i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
```

4.8. Funciones Closures

Los clausures son funciones que pueden tener otras funciones dentro de ellas.

JavaScript **otorga concesiones a la función interna**, **acceso total a todas las variables y funciones definidas dentro de la función externa** (y a todas las variables y funciones a las cuales la función externa tiene acceso).

Sin embargo, *la función externa no tiene acceso a las variables y funciones de la función interna*.

Además, *ya que la función interna tiene acceso al ámbito de la función externa*, ***las variables y funciones definidas en la función externa vivirán más tiempo que la función externa en sí misma*** **si la función interna las administra para sobrevivir más allá del ciclo de vida de la función externa**.

Una clausura es creada cuando la función interna es, de alguna manera, hecha disponible a cualquier ámbito fuera de la función externa. Y para acceder desde el exterior a las funciones internas, tenemos que invocarlas con el operador doble paréntesis ()().

Veamos un ejemplo

```
var a = "HOLA";
function global(){
    var b = "BUENOS";
    function local(){
        var c = "DIAS";
        return a + b + c;
    }
    return local;
}
global(); // Devuelve la función local: "function local() { var c
="DIAS"..""
global()(); // Devuelve la ejecución de la función local: "HOLA BUENOS
DIAS"
var closure = global();
closure(); // Devuelve lo mismo que global()(): "HOLA BUENOS DIAS"
```

Vistos estos conceptos ya podemos definir lo que es un closure.
Ejemplo

```
function suma(a){
    return function(b){
        return a+b;
    }
}
console.log(suma(1)(2));
```

Ejemplo

```
function getContador() {
    var contador=0;
    return function(){
        return contador++;
    }
}
var conta=getContador();
console.log(conta());
console.log(conta());
console.log(conta());
```

Veamos un ejemplo de closure con la siguiente función

```
var miContador = (function () {
       var _contador = 0; //En vars "privadas" se llama con `_`
       function incrementar () {
                return _contador++;
       }
       function decrementar () {
       return _contador--;
       }
       function valor () {
                return _contador;
       }
       return {
                incrementar: incrementar,
                decrementar: decrementar,
                valor: valor
       }
})();
miContador.valor(); // 0
miContador.incrementar();
miContador.incrementar();
miContador.valor(); // 2
miContador.decrementar();
miContador.valor(); // 1
```

Una propiedad seria _contador y los métodos serian: incrementar, decrementar y valor, todos ellos no son accesibles de fuera de la función. Para que se accesible se agrega en return.

5. Cadenas (String)

Las cadenas son textos que podemos usar en Javascript el tipo de datos es String. Es un conjunto de "elementos" de valores enteros de 16-bit no signados. Cada elemento en la cadena ocupa una posición en ella.

El primer elemento se encuentra en el índice 0, el siguiente en el índice 1, y asi sucesivamente. La longitud de una cadena es el número de elementos en ella. Es posible crear cadenas utilizando literales de cadena u objetos de cadena.

5.1. Literales de cadena

Es posible crear cadenas simples utilizando tanto comillas simples como dobles:

```
'foo'
"bar"
```

5.2. Secuencias de escape hexadecimal

El número despues de \x es interpretado como un número hexadecimal.

```
'\xA9' // "©"
```

5.3. Secuencias de escape Unicode

Las secuencias de escape Unicode requieren al menos cuatro dígitos hexadecimales después de \u.

```
'\u00A9' // "©"
```

5.4. Objetos cadena

El objeto String es un envoltorio alrededor del tipo de datos de cadena original.

```
var s = new String("foo"); // Crea un objeto String
console.log(s); // Muestra: { '0': 'f', '1': 'o', '2': 'o'}
typeof s; // Devuelve 'object'
```

5.5. Métodos de String

Método	Descripción
charAt, charCodeAt, codePointAt	Devuelve el caracter o el código del caracter en la posición especificada en la cadena.
indexOf, lastIndexOf	Devuelve la posición de la subcadena en la cadena o la última posición de una subcadena especificada respectivamente.
startsWith, endsWith, includes	Devuelve si la cadena empieza, termina o contiene una cadena especificada, o no.
concat	Combina el texto de dos cadenas y retorna una nueva cadena.
fromCharCode, fromCodePoint	Construye una cadena desde la secuencia de valores Unicode especificada. Este es un método de la clase String, no una instancia de String.
split	Divide un objeto String en un array de strings separados por substrings.
slice	Extrae una sección de un string y devuelve un nuevo string.
substring, substr	Devuelve un substring del string, bien especificando el comienzo y el final, o bien el índice inicial y la longitud del substring.
match, replace, search	Para trabajar con expresiones regulares
toLowerCase, toUpperCase	Devuelve el string en mayúsculas o en minúsculas
normalize	Devuelve es string Normalizado en Unicode.
repeat	Devuelve un string formado por los elementos del objetos repetidos el número de veces que le indiquemos.
trim	Elimina los espacios en blanco del principio y del final del string.

5.6. Cadenas de plantillas multilínea

Son cadenas literales que permiten expresiones incrustadas. Se pueden utilizar cadenas multilínea y funciones de interpolación de cadenas.

Se utiliza el carácter (` `) (acento grave) en lugar de las comillas dobles o simples.

```
(` `)  // Codigo ASCII Alt+96
```

Las plantillas de cadenas de texto pueden contener marcadores (placeholders). Son indicados por el signo de moneda y llaves (${expression}).

Ejemplo

```
Var nombre=”Jaime”;
console.log(`Bienvenido: ${nombre}`);
```

6. ARRAYS, NUMBER Y MODULOS JAVASCRIPT

6.1. Number

Es la clase del tipo primitivo number. Se codifican en formato de coma flotante con doble precisión (Es decir, con 64 bits / 8 bytes) y podemos representar números enteros, decimales, hexadecimales, y en coma flotante.

Declarar un Number

Para crear un número podemos hacerlo con la forma primitiva o con la clase Number . Por simplicidad se utiliza la forma primitiva.

```
var numero = 6;
var numero = new Number(6);
```

Infinity

Incluye los números Infinity y -Infinity para representar números muy grandes:

```
var num = Infinity;
var num2 = -Infinity;
```

NaN

NaN (Not A Number) para indicar que un determinado valor no representa un número:

```
num="a"/15;            //= NaN
```

Funciones de Number

JavaScript tiene funciones interesantes.

Función	Descripción	Ejemplo
parseInt(string, [b])	Convierte un string a número decimal Si agrega base, convierte en base b.	parseInt("1111"); // Devuelve 1111
parseFloat(string, [b])	Convierte un string a número float Si agrega base, convierte en base b.	parseFloat("5e3"); // Devuelve 5000

number.toFixed(x)	Redondea number con X decimales y lo convierte a string.	var n = 2.5674;
		n.toFixed(0);
		// Devuelve "3"

number.toExponential(x)	Redondea number float con X decimales y lo convierte a string.	var n = 2.5674;
		n.toExponential(2);
		// Devuelve "2.56e+0"

number.toString(b)	Convierte number a un string con base b	(15).toString(2); // Devuelve "1111"

6.2. Módulo Math

Math es una clase propia de JavaScript que contiene varios valores y funciones que nos permiten realizar operaciones matemáticas. Estos son los más utilizados:

Funcion Matematicas	Valor devuelto
Math.PI;	// Número Pi = 3.14159265...
Math.E;	// Número e = 2.7182818...
Math.random();	// Número aleatorio entre 0 y 1, ej: 0.45673858
Math.pow(2,6);	// Potencia de 2 elevado a 6 = 64;
Math.sqrt(4);	// raiz cuadrada de 4 = 2
Math.min(4,3,1);	// Devuelve el numero mínimo = 1
Math.max(4,3,1);	// Devuelve el numero máximo = 4
Math.floor(6.4);	// Devuelve el entero más próxima por debajo,
Math.ceil(6.4);	// Devuelve el entero más próxima por encima,
Math.round(6.4);	// Redondea a la parte entera más próxima,
Math.abs(x);	// Devuelve el valor absoluto de un número

Funcion Trigonometricas	Valor devuelto
Math.sin(x);	// Función seno de un valor
Math.cos(x);	// Función coseno de un valor
Math.tan(x);	// Función tangente de un valor
Math.log(x);	// Función logaritmo

6.3. Clase String

Un string es similar a un Array, con índices que van desde el 0 para el primer carácter hasta el último. Pero no es un array.

Declaración de String:

```
var texto = "Hola Jaime";
```

o

```
var texto = new String("Hola Jaime");
```

Los strings es una clase en JavaScript y tienen sus métodos.

Propiedades y Metodos	Valor devuelto
"Aplicaciones"[4]	// Devuelve "c"
Aplicaciones".length	// Devuelve la longitud de un string, es propiedad y debe ir sin paréntesis de función.
"Aplicaciones".charCodeAt(4));	// Devuelve el carácter 4 en formato UNICODE es decir de "c", el 99.
"Aplicaciones".indexOf("cion"));	// Devuelve el índice donde comienza el string "cion", el 6
"Aplicaciones".substring(4,8));	// Devuelve parte del string desde la posici'on 4 hasta la posicion (8-1) es decir "caci"

"lun mar mie jue vie".split(" ");	Convierte un string en array. Cada palabra debe estar separado por un semarador en común como un espacio.
	var miArray = "lun mar mie jue vie".split(" ");
	console.log(miArray);
	//Devuuelve ['lun', 'mar', 'mie', 'jue', 'vie']

6.4. Array

Es una colección de datos que pueden contener números, strings, objetos, otros arrays, etc...

DECLARAR UN ARRAY

Todo Array empieza con corchetes [...]

```
var miArray = [];
```

o

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

Ejemplos:

```
var miArray = [1, 2, 3, 4]; // Array de números
var miArray = ["Hola", "que", "tal"]; // Array de Strings
var miArray = [ {propiedad: "valor1" }, { propiedad: "valor2" }];

// Array de objetos
var miArray = [[2, 4], [3, 6]]; // Array de arrays, (Matriz);
var miArray = [1, true, [3,2], "Hola", {clave: "valor"}]; // Array mixto
```

ACCEDER A UN ELEMENTO DEL ARRAY

Se puede acceder a los elementos del array a través de su índice y con length conocer su longitud.

```
var miArray = ["uno", "dos", "tres"];
miArray[1]; // Devuelve: "dos"
miArray.length; // Devuelve 3
```

Si accedemos a una posición que no existe en el array, nos devuelve undefined .

```
miArray[8]; // undefined
```

Array de Objetos

```
var miArray=[
    {id: 5, nombre: 'Jaime', telefono: '4545545'},
    {id: 6, nombre: 'Juan', telefono: '3335545'},
    {id: 7, nombre: 'Jose', telefono: '4445545'}
];
console.log(miArray[5]['nombre']);
```

Métodos

Array es una clase de JavaScript, y heredan métodos:

Métodos	Valor Devuelto
Si tenemos: **var miArray = [3, 6, 1, 4];**	
miArray.sort();	// Devuelve un nuevo array ordenados: [1, 3, 4, 6]
miArray.pop();	// Devuelve el último elemento del array y lo saca. Devuelve 6 y miArray queda : [1, 3, 4]
miArray.push(2);	// Inserta un nuevo elemento en el array, devuelve la nueva longitud del array y queda : [1, 3, 4, 2]
miArray.reverse();	// Invierte el array, [2, 4, 3, 1]

miArray.join("x")	// Une todos los elementos en uno. X carácter de separación. Devuellve "2x4x3x1"
	var valor = 3;
	var template = [" ", valor, " "].join("");
	console.log(template); // Devuelve " 3 "

miArray.map(función)	// Ejecuta una función para todos los elementos
	var miArray = [2, 4, 6, 8];
	var resultados = miArray.map(function(elemento) {
	return elemento *= 2;
	});
	// resultados: [4, 8, 12, 16]

miArray.filter(funcion)	// Devuelve nuevo array con elementos filtrados por condición devuelta por función.
	var miArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
	var resultado = miArray.filter(function(elemento) {
	return elemento === 6;
	});
	console.log(resultado);

miArray.slice(posicion1,posicion2)	// Devuelve un array desde posicion1 hasta uno antes posicion2. Si omite posicion2 será hasta el final
	var miArray = [4, 8, 15, 16, 23, 42];
	// 0 1 2 3 4 5
	miArray.slice(2, 4); // [15, 16]

6.5. Desestructuración de Arrays

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// expected output: 10

console.log(b);
// expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// expected output: Array [30,40,50]
```

7. OBJETOS Y CLASES EN JAVASCRIPT

7.1. Objetos

Los objetos en JavaScript son simples colecciones de pares nombre-valor.

Como tales, son similares a:

- Diccionarios en Python
- Hashes en Perl y Ruby
- Tablas de Hash en C y C++
- Mapas de Hash en Java
- Matrices asociativas en PHP

El hecho esta simple estructura de datos sea tan ampliamente usada por su versatilidad. Todo (con excepción de los tipos primitivos) en JavaScript es un objeto, que cualquier programa JavaScript involucra naturalmente un gran manejo de búsquedas en tablas de hash.

La parte "name" es una cadena JavaScript, mientras que el valor puede ser cualquier valor, incluyendo más objetos.

Hay dos formas básicas de crear un objeto:

```
var obj = new Object({ propiedad: "valor" });

y:

var obj = { propiedad: "valor" };
```

```
var libroJaime = {
    titulo: 'Modelado de Software',
    autor: 'Jaime Suasnábar Terrel',
    paginas: 64,
    formatos: ["PDF", "ePub", "Mobi"],
    precio: 20.00,
    publicado: true
};
```

Devolver una Propiedad

Dos formas: notación punto o con la notación array:

```
libro.titulo; // Modelado de Software
libro['paginas']; // 64
```

Con la notación array, podemos acceder a las propiedades con variables.

Ejemplo:

```
var propiedad = "autor";
libro[propiedad]; // "Jaime Suasnabar"
```

Asignar una Propiedad

Dos formas: notación punto o con la notación array:

```
libro.precio = 25;
libro['publicado'] = true;
```

Objetos Anidados

La propiedad puede ser otro objeto y podemos acceder de la siguiente manera:

```
var libro = {
    titulo: "Modelado de Software",
    autor: {
        nombre: "Jaime Suasnabar",
        nacionalidad: "Peruano",
        edad: 40,
        contacto: {
        email: "jaime@gmail.com",
        twitter: "@jsuasnabar"
    }
    },
    editorial: {
        nombre: "Ibero Americana",
        web: "https://iberoamericana.com"
    }
};
```

Podemos acceder con notación punto, array, o mixto.

```
libro.autor.nombre; // "Jaime Suasnabar"
libro['autor']['edad']; // 40
libro['editorial'].web; // "https://iberoamericana.com"
libro.autor['contacto'].twitter; // "@jsuasnabar
```

Herencia de objetos

El siguiente ejemplo crea un objeto prototipo, Persona, y una instancia de ese prototipo, Tu.

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

// Definir un objeto
var Tu = new Persona("Tu", 24);
// Estamos creando una nueva persona llamada "Tu"
// (que fue el primer parametro, y su edad, el segundo)
```

Una vez creadas las propiedades del objeto, puede accederse a ellas de dos formas:

```
Tu.nombre = "Simon";
var nombre = obj.nombre;
```

y...

```
Tu["nombre"] = "Simon";
var nombre = obj["nombre"];
```

Desestructuración de objetos

La sintaxis de **desestructuración** es una expresión de JavaScript que permite desempacar valores de arreglos o propiedades de objetos en distintas variables.

```
let auto1 = {
    fabricante : "Toyota",
    modelo : "RAV4",
    color : "Rojo",
    cilindrada : 2.0,
}
let {fabricante, modelo} = auto1;
const {fabricante, modelo} = auto1;
```

8. CLASES EN ECMASCRIPT 6

Con la llegada de la nueva versión del estándar de JavaScript (ECMAScript 6 o ECMAScript 2015) la definición de una función como clase ha cambiado. ES6 aporta un azúcar sintáctico para declarar una clase como en la mayoría de los lenguajes de programación orientados a objetos, pero por debajo sigue siendo una función prototipal. El ejemplo anterior del Inventario, transformado a ES6 sería tal que así

```
class Inventario {
    constructor(nombre) {
              this.nombre = nombre;
              this.articulos = [];
    }
    add (nombre, cantidad) {
              this.articulos[nombre] = cantidad;
    }
    borrar (nombre) {
              delete this.articulos[nombre]
    }
    cantidad (nombre) {
              return this.articulos[nombre]
    }
    getNombre () {
              return this.nombre;
    }
}
```

Utilizando la palabra reservada class creamos una clase que sustituye a la función prototipal de la versión anterior.

El método especial constructor sería el que se definía en la función constructora anterior. Después los métodos add , borrar , cantidad y getNombre estarían dentro de la clase y sustituirían a las funciones prototipales de la versión ES5.

Su utilización es igual que en la versión anterior

```
var libros = new Inventario("Libros");
libros.add("AngularJS", 3);
libros.add("JavaScript", 10);
libros.add("NodeJS", 5);
libros.cantidad("AngularJS"); // 3
libros.cantidad("JavaScript"); // 10
libros.borrar("JavaScript");
libros.cantidad("JavaScript"); // undefined
```

Con esta nueva sintáxis podemos implementar herencia de una forma muy sencilla. Imagina que tienes una clase Vehículo de la siguiente manera:

```
class Vehiculo {
    constructor (tipo, nombre, ruedas) {
              this.tipo = tipo;
              this.nombre = nombre;
              this.ruedas = ruedas
    }
    getRuedas () {
              return this.ruedas
    }
    arrancar () {
              console.log(`Arrancando el ${this.nombre}`)
    }
    aparcar () {
              console.log(`Aparcando el ${this.nombre}`)
    }
}
```

Y quieres crear ahora una clase Coche que herede de vehículo para poder utilizar los métodos que esta tiene. Esto lo podemos hacer con la clase reservada extends y con super() llamamos al constructor de la clase que hereda

```
class Coche extends Vehiculo {
    constructor (nombre) {
              super('coche', nombre, 4)
    }
}
```

Si ahora creamos un nuevo objeto Coche podemos utilizar los métodos de la clase Vehiculo

```
let fordFocus = new Coche('Ford Focus')
fordFocus.getRuedas() // 4
fordFocus.arrancar() // Arrancando el Ford Focus
```

9. Colecciones con Llave

Este capítulo introduce las colecciones de datos las cuales son ordenadas mediante una clave; los objetos Map y Set contienen elementos cuya iteración se da conforme al orden de inserción.

9.1. El objeto Map

ECMAScript 6 introduce una nueva estructura de datos para mapear valores a valores. Un objeto Map es un simple mapeo clave/valor y puede iterar sobre sus elementos en el orden de inserción.

Puedes usar un **bucle for...of** para devolver un array de **[clave, valor]** en cada iteración.

El siguiente código muestra algunas operaciones básicas con un Map.

```
var sonidos = new Map();
sonidos.set("perro", "guau");
sonidos.set("gato", "miau");
sonidos.set("oveja", "beee");
sonidos.size; // 3
sonidos.get("zorro"); // undefined
sonidos.has("pájaro"); // false
sonidos.delete("perro");

for (var [clave, valor] of sonidos) {
  console.log(clave + " hace " + valor);
}
// "gato hace miau"
// "oveja hace beee"
```

9.2. Comparación de Object y Map

Tradicionalmente, objects ha sido usado para mapear strings a valores. Los objetos te permiten establecer claves con valores, recuperar esos valores, borrar las claves, y detectar si algo es almacenado como una clave. Los objetos Map, sin embargo, poseen ciertas ventajas sobre los objectos Object que los hacen mejores.

- Las claves de un Object son Strings, mientras que en un Map las claves pueden ser de cualquier tipo.
- Se puede obtener el tamaño de un Map fácilmente mientras en un Object se debe realizar un registro manual para obtenerlo.
- La iteración sobre mapas se produce en el orden en que se insertaron los elementos en el mapa.
- Un Object tiene un prototipo, por eso existen claves por defecto en el map. Esto puede evitarse haciendo map = Object.create(null)).

Estos dos consejos pueden guiarte a la hora de decidir si usar un Map o un Object:

- Usa mapas preferiblemente cuando las claves son desconocidas hasta el tiempo de ejecución, o cuando todas las claves son del mismo tipo y todos los valores son del mismo tipo.
- Usa objetos cuando sea lógico operar sobre elementos individuales.

Test de la Primera Actividad

Pregunta 1:

En JavaScript, ¿cómo se escribe una sentencia condicional para comprobar que la variable "i" es igual a 5?

a) if i=5 then

b) if(i=5)

c) if i==5 then

d) if(i==5)

Pregunta 2:

¿Un fichero externo con código JavaScript debe contener la etiqueta <script>?
a) Sí

b) No

Pregunta 3:

¿Cómo se hace en CSS para que cada palabra en un texto comience con una letra en mayúsculas?
a) text-transform:capitalize

b) text-transform:uppercase

c) text-transform:first-letter

d) No se puede hacer con CSS

Pregunta 4:

Si a = true y b = false, la expresión:
!(!(a || b) && !(a))
Toma el mismo resultado que:
a) a && b

b) !(a || !b)

c) b ll (a&&b)

d) !(!a || b) || !b

Pregunta 5:

En HTML, la etiqueta utilizada para agrupar campos en un formulario es
a) <fieldset>

b) <legend>

c) <label>

d) Las anteriores respuestas no son correctas

Pregunta 6:

En JavaScript, ¿cómo se calcula el máximo de los números 2 y 4?
a) ceil(2, 4)

b) top(2, 4)

c) Math.ceil(2, 4)

d) Math.max(2, 4)

Pregunta 7:

En JavaScript, el operador para concatenar cadenas es
a) "&"
b) "+"
c) "."
d) Las anteriores respuestas no son correctas

Pregunta 8:

En JavaScript, ¿cómo se abre una ventana nueva del navegador?
a) document.open("pagina.html")
b) document.new("pagina.html")
c) window.open("pagina.html")
d) window.new("pagina.html")

Pregunta 9:

En JavaScript, ¿cómo se inserta un comentario que ocupa una línea?
a) <!-- Comentario -->
b) // Comentario
c) ' Comentario
d) Las anteriores respuestas no son correctas

Pregunta 10:

En JavaScript, ¿cómo se muestra una ventana con el mensaje "Hola mundo!"?

a) alert("Hola mundo!");

b) alertBox = "Hola mundo!";

c) alertBox("Hola mundo!);

d) msgBox("Hola mundo!);

Pregunta 11:

En un formulario HTML, cuando se pulsa sobre un botón de tipo submit, los datos introducidos en el formulario se envían a la URL indicada en el atributo del formulario llamado
a) method

b) post

c) target

d) Las anteriores respuestas no son correctas

Pregunta 12:

¿Cuál de las siguientes afirmaciones sobre XML es correcta?
a) XML es una extensión multidocumento de HTML

b) XML es un metalenguaje que se emplea para definir otros lenguajes

c) XML es la versión ligera de XHTML

d) Las anteriores respuestas no son correctas

Pregunta 13:

En JavaScript, ¿cuál es la forma correcta de escribir la cadena "Hola mundo" en una página web?
a) document.print("Hola mundo")

b) document.write("Hola mundo")

c) window.print("Hola mundo")

d) window.write("Hola mundo")

Pregunta 14:

¿Qué afirmación no es correcta respecto al lenguaje XML?
a) Los documentos HTML son directamente compatibles con las reglas de XML

b) XML permite definir nuevas etiquetas y atributos

c) Los documentos XML pueden ser validados para comprobar si son correctos

d) En XML la estructura del documento puede anidarse en varios niveles de complejidad

Pregunta 15:

Cuando se emplea el método "post" de envío de un formulario, ¿los datos del formulario se muestran en la URL?
a) Siempre

b) Sí, si así se ha indicado en el formulario

c) Nunca

Pregunta 16:

En XHTML, para definir una línea horizontal de separación se emplea
a) <hr />

b) <line />

c) <sep />

d) Las anteriores respuestas no son correctas

Pregunta 17:

En JavaScript, ¿qué función se emplea para convertir una cadena a minúsculas?
a) lower()

b) lowerCase()

c) toLower()

d) toLowerCase()