Desarrollo Web en Entorno Cliente

Tema 2

APLICACIÓN Y VERIFICACION DE LA SINTAXIS DEL LENGUAJE

Tema 2 Aplicación y verificación de la sintaxis de JavaScript

Tema 2

1. Selección del lenguaje de programación de clientes web
2. Tipos de datos : asignaciones expresiones
3. Introducción a las funciones
4. Introducción a los objetos en JavaScript
5. Variables y ámbitos de utilización
6. Eventos
7. Objetos String o cadenas de caracteres
8. Números
9. Fechas
10. Arrays
11. Sentencias condicionales y bubles

1.- Selección de lenguaje de programación en clientes web

- La decisión de elegir el lenguaje a utilizar para la realización de nuestro portal web es de vital importancia
- JavaScript fue desarrollado por la empresa Netscape, con el nombre inicial de Mocha. Seguidamente pasó a llamarse LiveScript, hasta que cambió para adoptar el nombre actual, JavaScript.
- JavaScript se diseñó teniendo como base el lenguaje C.
  - Todos los lenguajes de alto nivel de la programación moderna tienen la misma base.
  - Esto supone la facilidad de adoptar, convertir y compatibilizar JavaScript con otros lenguajes de la misma base.

Cualquier programa de JavaScript puede comunicarse con la consola del navegador mediante la sentencia:

```
console.log("información para la consola");
```

En la consola, se puede escribir literales y también valores de variables. Todos los navegadores tienen consola y es muy utilizada por los programadores con fines de depuración.

1.- Selección de lenguaje de programación en clientes web

LAS 10 REGLAS BÁSICAS DE LA SINTAXIS DE JAVASCRIPT

1. Las instrucciones en JavaScript terminan en un punto y coma.
2. Los números que tengan decimales utilizaran el punto como separador de las unidades con la parte decimal.

```
var pi = 3.14;
```

3. Los literales se pueden escribir entre comillas simples o dobles.

```
var s1 = "Hola";
var s2 = 'Hola';
```

4. Cuando sea necesario declarar una variable, se utilizará la palabra reservada var.
5. El operador de asignación, al igual que en la mayoría de lenguajes, es el símbolo igual (=).

LAS 10 REGLAS BÁSICAS DE LA SINTAXIS DE JAVASCRIPT

6. Se pueden utilizar los siguientes operadores aritméticos: (+ - * /). Ejemplo:

```
var x = (5*4)/2+1;
```

7. En las primeras expresiones, también se pueden utilizar variables. Ejemplo:

```
var t = 4;
var x = (5*t)/2+1;
var y;
y = x*2;
```

LAS 10 REGLAS BÁSICAS DE LA SINTAXIS DE JAVASCRIPT

8. Comentarios en JavaScript. Existen dos opciones para comentar el código:

   //Cuando se desea comentar el resto de la línea a partir de estas dos barras invertidas.

   /* y */ para comentar todo el contenido entre ambas etiquetas.

9. Los identificadores en JavaScript comienzan por una letra o la barra baja (_) o el símbolo del dólar ($).
10. JavaScript es sensible a las mayúsculas y minúsculas (case-sensitive).

PALABRAS RESERVADAS EN JAVASCRIPT

- JavaScript contiene una serie de palabras que no podemos utilizar para definir nombres de variables, funciones o etiquetas.
- Según la versión 6 de ECMAScript, las palabras reservadas son las que vemos a continuación:

PALABRAS RESERVADAS				
break	default	for	new	typeof
case	delete	function	return	var
class	do	if	super	void
catch	else	import	switch	while
const	export	in	this	with
continue	extends	instanceof	throw	yield
debugger	finally	let	try	

1.- Selección de lenguaje de programación en clientes web

- Las siguientes palabras están reservadas como palabras futuras cuando se encuentre el modo correcto para su estructura de código:

PALABRAS RESERVADAS				
enum	implements	package	protected	static
interface	private	public		

1.- Selección de lenguaje de programación en clientes web

- Las siguientes palabras están reservadas como palabras futuras por la antigua especificación ECMAScript1 hasta ECMAScript3. De todas formas, para una buena praxis de programación, las valoraremos como palabras reservadas:

PALABRAS RESERVADAS				
abstract	boolean	byte	char	double
final	float	goto	int	long
native	short	synchronized	transient	volatile

- Adicionalmente, los literales null, true, y false están reservadas en ECMAScript para usos normales.

1.- Selección de lenguaje de programación en clientes web

VARIABLES

- El elemento más básico para trabajar en programación es la variable.
- Definimos variable como un espacio de memoria para almacenar información.
- El tiempo que dicha información permanece almacenada, dependerá del desarrollador y del tipo de memoria que estemos utilizando.
  - En el momento que el navegador limpia la ventana, cualquier variable conocida será eliminada.
- El primer paso para hacer uso de una variable es la creación o declaración en el código.

1.- Selección de lenguaje de programación en clientes web

- El primer paso para hacer uso de una variable es la creación o declaración en el código. Por tanto, podemos realizar esta operación de dos formas diferentes:
  - Mediante la palabra reservada **var** seguida del nombre de la variable o identificador.
  - Por ejemplo, para declarar una variable "numero", el código de JavaScript será:

    var numero;
  - Como complemento, además de crear la variable, también podemos asignar un valor durante la operación, por ejemplo:

    var numero = 14;
- En este caso, en la declaración de la variable no hay que indicar el tipo de datos que va a contener.
  - Por esta razón, una variable de JavaScript podrá almacenar diferentes tipos de valores, lo que supone una ventaja, ya que no obliga al programador a fijar el tipo de datos a utilizar.

1.- Selección de lenguaje de programación en clientes web

Hoisting (elevación) en las variables en JavaScript.

- El **hoisting** es una característica del motor de JavaScript que "levanta" las declaraciones de variables y funciones **al inicio** de su ámbito antes de ejecutar el código.
- "Levantar" las declaraciones de variables al inicio de su ámbito significa que el motor de JavaScript **procesa primero** todas las declaraciones de variables y funciones **antes de ejecutar** cualquier parte del código.

```
function exampleWithLet() {
    console.log(x); // ReferenceError: Cannot access 'x' before initialization
    let x = 10;
    console.log(x); // 10
}

exampleWithLet();
```

1.- Selección de lenguaje de programación en clientes web

Tipos de variable

- Se puede declarar variables usando tres palabras claves, cada una con sus características propias:
  - **var**
    - Ámbito de función
  - **let**
    - Ámbito de bloque
  - **const**
    - Definición de constante

1.- Selección de lenguaje de programación en clientes web

Tipos de variable Var

- Las variables declaradas con var tienen un alcance de función, lo que significa que son accesibles dentro de la función en la que fueron declaradas.

```
function ejemplo() {
    var mensaje = "Hola, mundo!";
    console.log(mensaje); // "Hola, mundo!"
}
console.log(mensaje); // ReferenceError: mensaje is not defined
```

- Si se declaran fuera de cualquier función, tienen un alcance global.

```
var mensaje = "Hola, mundo!";

function saludar() {
  console.log(mensaje); // Imprime: Hola, mundo!
}

saludar();
```

Redeclarar una variable (var)

Comportamiento de la redeclaración con var:

- No afecta el valor:
  - Si redeclaras una variable con var y le asignas un nuevo valor, este nuevo valor sobrescribirá el valor anterior.
- Mantiene el ámbito:
  - La redeclaración no cambia el ámbito de la variable.
  - Si una variable fue declarada globalmente, seguirá siendo global incluso si la redeclaras dentro de una función.

Se desaconseja la redeclaración con var

- **Confusión**:
  - La redeclaración puede hacer que el código sea más difícil de entender y mantener.
- **Errores**:
  - La redeclaración accidental puede sobrescribir valores importantes y causar errores difíciles de depurar.

1.- Selección de lenguaje de programación en clientes web

Hoisting (elevación) en las variables var en JavaScript.

Temporal Dead Zone (TDZ)

- Es el período entre el inicio del ámbito de la variable (el punto donde se declara) y el punto donde se inicializa.
- Durante este tiempo, cualquier acceso a la variable causará un **ReferenceError.**

Redeclarar una variable

- Implica volver a declarar una variable con el mismo nombre dentro del mismo ámbito

```
var x = 10;
console.log(x); // Imprime 10

var x = 20; // Redeclaración
console.log(x); // Imprime 20
```

1.- Selección de lenguaje de programación en clientes web

Alternativas hay a var

- **let**:
  - Con let, no se permite la redeclaración dentro del mismo ámbito. Si intentas redeclarar una variable con let, obtendrás un error de sintaxis.
- **const**:
  - Con const, no se permite la redeclaración y tampoco se puede reasignar el valor de la variable una vez inicializada.

1.- Selección de lenguaje de programación en clientes web

Alcance de las variables var dentro de bucles y condicionales

- Las variables declaradas con var en JavaScript tienen un alcance funcional (no de bloque como let),
  - Esto significa que son accesibles desde cualquier parte de la función en la que se declaran.
- Consecuencias:
  - Bucles:
    - Una variable con var dentro de un bucle, será accesible desde cualquier parte de la función, incluso después de que el bucle haya terminado.
    - Esto puede llevar a problemas si no tienes cuidado, ya que la variable puede mantener el último valor asignado dentro del bucle.
  - Condicionales:
    - Lo mismo aplica para los condicionales. de un if, else if o else, esa variable será accesible desde cualquier parte de la función.

1.- Selección de lenguaje de programación en clientes web

Resumen de las características de var:

- **Alcance de función**: Las variables var son accesibles dentro de la función en la que se declaran.
- **Hoisting**: Las declaraciones de var se elevan al comienzo de su contexto de ejecución.
- **Redeclaración** permitida: Las variables var pueden ser redeclaradas en el mismo ámbito.
- **Sin alcance de bloque**: Las variables var no están limitadas a bloques de código como bucles o condicionales.

1.- Selección de lenguaje de programación en clientes web

let

- Se utiliza para declarar variables con alcance de bloque.
  - Esto significa que la variable **solo es accesible** dentro del bloque donde fue declarada.
  - Un bloque está definido por llaves {}.

```
{
  let mensaje = "Hola desde el bloque";
  console.log(mensaje); // Imprime: Hola desde el bloque
}

console.log(mensaje); // Error: mensaje no está definida
```

1.- Selección de lenguaje de programación en clientes web

let

- Importante: el alcance de bloque
  - **Evita conflictos de nombres**:
    - Al limitar el alcance de una variable a un bloque específico, reduces la probabilidad de que se produzcan conflictos de nombres con otras variables que tengan el mismo nombre en otros bloques.
  - **Hace el código más legible**:
    - El alcance de bloque hace que el código sea más fácil de entender, ya que puedes ver claramente dónde se utiliza cada variable.
  - **Mejora la modularidad**:
    - Al limitar el alcance de las variables, puedes crear módulos de código más independientes y reutilizables.

1.- Selección de lenguaje de programación en clientes web

let :

- **Bucles**:
  - Dentro de un bucle for, puedes declarar una variable con let para controlar la iteración sin que afecte a variables con el mismo nombre fuera del bucle.
- **Condicionales**:
  - Dentro de un if, else if o else, puedes declarar variables con let para realizar operaciones específicas dentro de esos bloques.
- **Funciones**:
  - Dentro de una función, puedes declarar variables con let para almacenar valores locales a esa función.

1.- Selección de lenguaje de programación en clientes web

Let : Hoisting en variables let

- Las variables declaradas con let **no** son hoisteadas.
- No se pueden usar antes de ser declaradas:
  - Si intentas acceder a una variable let antes de su declaración, obtendrás un error de referencia.
- No hay inicialización predeterminada:
  - A diferencia de var que se inicializa con **undefined**, las variables let no tienen un valor predeterminado antes de su asignación.

```
function test() {
  console.log(x); // ReferenceError: Cannot access 'x' before initialization
  let x = 10;
}
```

1.- Selección de lenguaje de programación en clientes web

Resumen let

- **Alcance de bloque**: let solo es accesible dentro del bloque donde se declaró.
- **Hoisting sin inicialización**: let se eleva pero no se inicializa, generando un error si se accede antes de su declaración.
- **Sin redeclaración** en el mismo ámbito: No se puede redeclarar una variable let en el mismo ámbito.
- **Permite reasignación**: Puedes cambiar el valor de una variable let después de haberla declarado.
- **Uso seguro en bucles**: Cada iteración del bucle tiene su propio ámbito.

1.- Selección de lenguaje de programación en clientes web

const

- Se utiliza para declarar una constante, es decir, una variable cuyo valor no puede ser cambiado después de su inicialización.
- Siempre deben ser inicializadas en el momento de su declaración.

```
// Inicialización correcta
const PI = 3.14159;
const nombre = "Juan";

// Inicialización incorrecta (genera un error)
const edad; // Falta la inicialización
```

1.- Selección de lenguaje de programación en clientes web

Operadores:

- Operadores lógicos y de comparación

OPERADORES DE COMPARACIÓN EN JAVASCRIPT			
Sintaxis	Nombre	Tipos de operandos	resultados
==	Igualdad	Todos	Boolean
!=	Distinto	Todos	Boolean
===	Igualdad estricta	Todos	Boolean
!==	Desigualdad estricta	Todos	Boolean
>	Mayor que	Todos	Boolean
>=	Mayor o igual que	Todos	Boolean
<	Menor que	Todos	Boolean
<=	Menor o igual que	Todos	Boolean

1.- Selección de lenguaje de programación en clientes web

Operadores:

```
30 == 30 // true
30 == 30.0 // true
5 != 8 // true
9 > 13 // false
7.29 <= 7.28 // false
```

```
"Marta" == "Marta" // true
"Marta" == "marta" // false
"Marta" > "marta" // false
"Mark" < "Marta" // true
```

```
"123" == 123 // true
```

- JavaScript cuando realiza esta comparación, convierte la cadena en su número correspondiente y luego realiza la comparación.
- También dispones de otra opción, que consiste en convertir mediante las funciones **parseInt**() o **parseFloat**() el operando correspondiente:
- Los operadores ==

```
parseInt("123") == 123 // true
```

el dato como el tipo de dato.
  - El operador === sólo devolverá true, cuando los dos operandos son del mismo tipo de datos (por ejemplo, ambos son números) y tienen el mismo valor.

1.- Selección de lenguaje de programación en clientes web

- Operadores Aritméticos

OPERADORES DE COMPARACIÓN EN JAVASCRIPT			
Sintaxis	Nombre	Tipos de operandos	Resultados
+	Más	integer, float, string	integer, float, string
-	Menos	integer, float	integer, float
*	Multiplicación	integer, float	integer, float
/	División	integer, float	integer, float
%	Módulo	integer, float	integer, float
++	Incremento	integer, float	integer, float
--	Decremento	integer, float	integer, float
+valor	Positivo	integer, float, string	integer, float, string
-valor	Negativo	integer, float, string	integer, float, string

1.- Selección de lenguaje de programación en clientes web

```
var a = 10; // Inicializamos a al valor 10
var z = 0; // Inicializamos z al valor 0
z = a; // a es igual a 10, por lo tanto, z es igual a 10.
z = ++a; // el valor de a se incrementa justo antes de ser asig-
nado a z, por lo que a es 11 y z valdrá 11.
z = a++; // se asigna el valor de a (11) a z y luego se incrementa el va-
lor de a (pasa a ser 12).
z = a++; // a vale 12 antes
```

```
var x = 2;
var y = 8;
var z = -x; // z es igual a -2, pero x sigue siendo igual a 2.
z = - (x + y); // z es igual a -10, x es igual a 2 e y es igual a 8.
z = -x + y; // z es igual a 6, pero x sigue siendo igual a 2 e y igual a 8.
```

1.- Selección de lenguaje de programación en clientes web

- Operadores de Asignación

OPERADORES DE ASIGNACIÓN EN JAVASCRIPT			
Sintaxis	Nombre	Ejemplo	Significado
=	Asignación	x = y	x = y
+=	Sumar un valor	x += y	x = x + y
-=	Substraer un valor	x -= y	x = x - y
*=	Multiplicar un valor	x *= y	x = x * y
/=	Dividir un valor	x /= y	x = x / y
%=	Módulo de un valor	x %= y	x = x % y
<<=	Desplazar bits a la izquierda	x <<= y	x = x << y
>=	Desplazar bits a la derecha	x >= y	x = x > y
>>=	Desplazar bits a la derecha rellenando con 0	x >>= y	x = x >> y
>>>=	Desplazar bits a la derecha	x >>>= y	x = x >>> y
&=	Operación AND bit a bit	x &= y	x = x & y
|=	Operación OR bit a bit	x |= y	x = x | y
^=	Operación XOR bit a bit	x ^ = y	x = x ^ y
[]=	Desestructurando asignaciones	[a,b]=[c,d]	a=c, b=d

1.- Selección de lenguaje de programación en clientes web

- Operadores booleanos
  - La mayor parte de la programación tiene un gran componente de lógica,
    - Por ello los operadores booleanos juegan un gran papel.
  - Los operadores booleanos te van a permitir evaluar expresiones, devolviendo como resultado true (verdadero) o false (falso).

OPERADORES BOOLEANOS EN JAVASCRIPT			
Sintaxis	Nombre	Operando	Resultados
&&	AND	Boolean	Boolean
||	OR	Boolean	Boolean
!	Not	Boolean	Boolean

1.- Selección de lenguaje de programación en clientes web

TABLA DE VALORES DE VERDAD DEL OPERADOR AND			
Operando izquierdo	Operador AND	Operando Derecho	Resultados
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

TABLA DE VALORES DE VERDAD DEL OPERADOR OR			
Operando izquierdo	Operador OR	Operando Derecho	Resultados
True	||	True	True
True	||	False	True
False	||	True	True
False	||	False	False

1.- Selección de lenguaje de programación en clientes web

```
!true // resultado = false
!(10 > 5) // resultado = false
!(10 < 5) // resultado = true
!("gato" == "pato") // resultado = true
5 > 1 && 50 > 10 // resultado = true
5 > 1 && 50 < 10 // resultado = false
5 < 1 && 50 > 10 // resultado = false
5 < 1 && 50 < 10 // resultado = false
```

```
5 > 1 || 50 > 10 // resultado = true
5 > 1 || 50 < 10 // resultado = true
5 < 1 || 50 > 10 // resultado = true
5 < 1 || 50 < 10 // resultado = false
```

1.- Selección de lenguaje de programación en clientes web

- Operadores de majeño de String
  - Se utilizarán los operadores + y += para concatenar strings. Véase el siguiente ejemplo de uso:

```
var saludo = "Hola" + " " + "mundo"
```

- Operadores de tipo
  - Los operadores de tipo permiten conocer si un objeto es una instancia de un tipo concreto o bien conocer el tipo de una variable.
  - Los operadores de tipo disponibles en JavaScript:
    - typeof. Devuelve el tipo de una variable.
    - instanceof. Devuelve true si un objeto es una instancia de un tipo de obje-to

2.- Tipos de datos, signaciones y expresiones

- Las variables en JavaScript podrán contener cualquier tipo de dato

TIPO	EJEMPLO
Cadena (String)	var nombre = "David";
Número (Number)	var edad = 25;
Booleano (Boolean)	true
Array	var asignatura = ["Lengua", "Matemáticas", "Ingles"];
Object	var persona = {nombre:"David", apellido: "Moreno"];

```
var miEntero = 12;
var miDecimales = 2.33;
var comaFlotante = 1234.9876555;
var miCadena = "Esto es una cadena!";
var miBoleano = true;
var miVariable = null;
```

2.- Tipos de datos, signaciones y expresiones

- Valor null
  - **null** para los lenguajes de programación es nada o algo que no existe.
    - Generalmente, cuando se asigna null a una variable, es porque es o será un objeto.

```
var persona = null;
persona = {nombre:"David", apellido:"Moreno"};
```

- Conversiones de tipos de datos
  - JavaScript intenta realizar la mejor conversión cuando realiza esas operaciones, pero a veces no es el tipo de conversión que interesa.

```
4 + 5 // resultado = 9
```

2.- Tipos de datos, signaciones y expresiones

- Si uno de esos números está en formato de cadena de texto, JavaScript lo que hará es intentar convertir el otro número a una cadena y los concatenará.

```
4 + "5" // resultado = "45"
```

```
4 + 5 + "6" // resultado = "96"
```

- La expresión se evalúa de izquierda a derecha.
  - La primera operación funciona correctamente devolviendo el valor de 9 pero al intentar sumarle una cadena de texto "6"
  - JavaScript lo que hace es convertir ese número a una cadena de texto y se lo concatenará al comienzo del "6".

- Para convertir cadenas a números dispones de las funciones: **parseInt**() y **parseFloat**().

```
parseInt("34") // resultado = 34
parseInt("89.76") // resultado = 89
```

```
parseFloat("34") // resultado = 34
parseFloat("89.76") // resultado = 89.76
4 + 5 + parseInt("6") // resultado = 15
```

2.- Tipos de datos, signaciones y expresiones

- Si lo que deseas es realizar la conversión de números a cadenas, simplemente es concatenar una cadena vacía al principio.
  - De esta forma el número será convertido a su cadena equivalente:

```
("" + 3400) // resultado = "3400"
("" + 3400).length // resultado = 4
```

- Es posible también convertir variables de tipo fecha a String con la función toDateString()

```
var fecha = new Date();
var cad = fecha.toDateString(); //cad contendrá la fecha actual
```

- Si se quiere convertir la hora en formato UTC, se puede utilizar alternativamente la función toUTCString()
  - Formato UTC: formato calculado a partir a partir del 1 enero de 1972

```
var fecha = new Date();
var cad = fecha.toUTCString(); //cad contendra la fecha actual en formato UTC
```

3.- Introducción a las funciones

- Las funciones son uno de los elementos de la base de la programación estructurada.
- Reutilizar el código es algo básico en cualquier lenguaje de programación.
- Un ejemplo de estructura básica de una función es el siguiente:

```
function suma(a, b){
    return a + b;
}
```

- La palabra reservada **function** para que JavaScript interprete que se encuentra frente a una función seguida de su nombre.
  - Esta función acepta dos parámetros a y b y devolverá (return) la suma de ambos.
  - Una función puede tener cero, uno o varios parámetros.
  - El código de la función está dentro de las llaves { y }.
  - La función devolverá el resultado de la sentencia return.

4.- Introducción a los objetos JavaScript

- Un objeto en JavaScript tiene características llamadas atributos.
- Cada atributo se verá reflejado en una variable dentro del objeto.
- Un objeto se diferencia de otro por el valor de sus atributos.

```
var perro = {
    raza: "Pastor Aleman",
    peso: 30,
    altura: 67,
    color: "marron"

}
```

- Se pueden encontrar definiciones de objetos en una sola línea:

```
var perro = {raza: "Pastor Aleman", peso: 30, altura: 67, color: "marron}
```

4.- Introducción a los objetos en JavaScript

- El acceso a los atributos se puede realizar de dos formas:
- Operador .

```
perro.raza;
```

- Operador corchetes : el nombre del atributo con comillas dobles

```
perro["raza"];
```

5.- Variables y ámbitos de utilización

- Variables globales y variables locales

```
var a; // Esta es una variable global
a = 10;

function suma(){
    var b = 5;
    return a + b; // la función devolvera 15
}
```

  - a : variable global que puede utilizarse en otras funciones
  - b : variable local que solo puede ser utilizada dentro de la función suma
- I**mportante**: Las variables y sus valores, desaparecen cuando se cierra la página web.

5.- Variables y ámbitos de utilización

```
a = 10;

function suma(){
    var b = 5;
    c = 20;
    return a + b; // la función devolvera 15
}
```

- Que sucede con la variable **c**?
  - Se considera variable global a toda la pagina

5.- Variables y ámbitos de utilización

- Diferencia entre var y let
- Una variable con var, la variable puede utilizarse fuera del bloque, donde fue declarada

```
var a = 10; // x vale 10
{
    let a = 5, // a vale 5
}
// aqui a vale 10
{
    let b = 3;
}
// aqui no se puede utilizar b
{
    var c = 7;
}
// aqui si se puede utilizar c y valdra 7
```

```
function varTest() {
    var x = 31;
    if (true) {
      var x = 71;  // ¡misma variable!
      console.log(x);  // El valor de x es 71
    }
    console.log(x);  // El valor de x es 71
  }

  function letTest() {
    let x = 31;
    if (true) {
      let x = 71;  // variable diferente
      console.log(x);  // El valor de x es 71
    }
    console.log(x);  // El valor de x es 31
  }
```

6.- Eventos

- Los eventos son cualquier suceso que le pueda ocurrir a un elemento HTML.
- JavaScript puede darse cuenta de ese evento y reaccionar a este ejecutando el código que se haya programado.
- Algunos eventos que pueden que pueden ocurrir en una página web son:
  - Pulsar un botón
  - Modificar un campo de texto.
  - Pulsar una tecla.
  - La página ha terminado de cargarse.
  - Hacer clic sobre un elemento HTML.

6.- Eventos

- El formato de programar un evento en JavaScript sería el siguiente:
  - **<Elemento_HTML evento='codigo_JavaScript'>**
- Es posible cambiar las comillas simples por comillas dobles.
  - Se pueden utilizar de forma indistinta.
- Ejemplo de un evento en JavaScript programado para un botón:

```
<button onclick="funcionAEjecutar()">Clic</button>
```

6.- Eventos

- Tipos de eventos

EVENTO	DESCRIPCIÓN	ELEMENTOS PARA LOS QUE ESTÁ DEFINIDO
onblur	Un elemento pierde el foco	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Un elemento ha sido modificado	<input>, <select>, <textarea>
onclick	Pulsar y soltar el ratón	Todos los elementos
ondblclick	Pulsar dos veces seguidas con el ratón	Todos los elementos
onfocus	Un elemento obtiene el foco	<button>, <input>, <label>, <select>, <textarea>, <body>

6.- Eventos

onkeydown	Pulsar una tecla y no soltarla	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	Página cargada completamente	<body>
onmousedown	Pulsar un botón del ratón y no soltarlo	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón "sale" del elemento	Todos los elementos
onmouseover	El ratón "entra" en el elemento	Todos los elementos
onmouseup	Soltar el botón del ratón	Todos los elementos
onreset	Inicializar el formulario	<form>
onresize	Modificar el tamaño de la ventana	<body>

6.- Eventos

onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>

6.- Eventos

LOS LISTENER

- Otra forma de añadir los eventos a los elementos u objetos HTML del DOM listeners.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Título de la WEB</title>
</head>
<body>
    <p id="myFPtext">texto a modificar</p>
    <button id="myFPboton">Dale fuerte!</button>
    <script>
        function dale(){
            var x = document.getElementById("myFPtext");
            x.innerHTML = "Hola";
        }
     document.getElementById("myFPboton").addEventListener("click", dale, false);

    </script>
</body>
</html>
```

6.- Eventos

- Se utiliza el método **addEventListener** para vincular un evento a una función JavaScript.
- Los parámetros de la función **addEventListener**() son los siguientes:
  1. El evento (sin el "on").
  2. La función por ejecutar.
  3. useCapture.
     - Se utiliza si el usuario quiere iniciar la captura para que todos los eventos de este tipo se lancen en el mismo listener.
     - Generalmente, se utiliza false en este parámetro.

7.- Objetos String : cadenas de texto

- Se utiliza para representar y manipular una secuencia de caracteres.
- Para utilizar las comillas dentro de un string, se necesitará utilizar secuencias de escape.
  - Un ejemplo de introducción de una barra invertida dentro de una cadena sería el siguiente:

```
var web = "la mejor \"web\" de tecnología";
```

- Es posible incluir tabulación (\t) y saltos de línea (\n) dentro de las cadenas de caracteres.

```
alert("Hola \tme llamo \nCarlos");
```

7.- Tipo Números

- Un valor numérico se puede transformar en string con la expresión: numero.toString()

```
var num = 15;
num.toString(); // devolvera 15 pero como cadena de caracteres
(888).toString(); // devolvera 888 como cadena de caracteres
(888 + 111).toString(); // // realiza la operación y devuelve 999 pero como ca-
dena de caracteres
```

- El método toFixed() formatea un número usando notación de punto fijo.
  - Sintaxis: numObj.toFixed([digitos])
  - Parámetros
    - Dígitos: Opcional. El número de dígitos que aparecen después del punto decimal; este puede ser un valor entre 0 y 20, inclusive, algunas implementaciones pueden soportar un rango más amplio de valores. Si el argumento es omitido, es tratado como 0.

7.- Tipo Números

- Ejemplo:

```
var numObj = 12345.6789;

numObj.toFixed();       // Returns '12346' Realiza redondeo
numObj.toFixed(1);      // Returns '12345.7' Realiza redondeo
numObj.toFixed(6);      // Returns '12345.678900' Añade ceros
2.34.toFixed(1);        // Returns '2.3' Realiza redondeo
2.35.toFixed(1);        // Returns '2.4' Realiza redondeo
-2.34.toFixed(1);       // Returns -2.3 Realiza redondeo negativo
(-2.34).toFixed(1);     // Returns '-2.3' Realiza redondeo negativo
```

7.- Tipo Número

- Ajuste de decimales
  - El método toFixed() formatea un número usando notación de punto fijo.
  - Sintaxis: numObj.toFixed([digitos])
  - Parámetros
    - Dígitos: Opcional.
      - El número de dígitos que aparecen después del punto decimal; este puede ser un valor entre 0 y 20, inclusive, algunas implementaciones pueden soportar un rango más amplio de valores.
      - Si el argumento es omitido, es tratado como 0.

```
var numObj = 12345.6789;

numObj.toFixed();      // Returns '12346' Realiza redondeo
numObj.toFixed(1);     // Returns '12345.7' Realiza redondeo
numObj.toFixed(6);     // Returns '12345.678900' Añade ceros
2.34.toFixed(1);       // Returns '2.3' Realiza redondeo
2.35.toFixed(1);       // Returns '2.4' Realiza redondeo
-2.34.toFixed(1);      // Returns -2.3 Realiza redondeo negativo
(-2.34).toFixed(1);    // Returns '-2.3' Realiza redondeo negativo
```

7.- Tipo Número

- Precición
  - En JavaScript, el método toPrecisión() servirá para ajustar la precisión de un número.
  - Sintaxis: numObj.toPrecision([precision])
  - Parámetros
    - Precisión. Opcional. Un entero que especifica el número de dígitos significativos

```
var numObj = 5.123456;

console.log(numObj.toPrecision());    // logs '5.123456'
console.log(numObj.toPrecision(1));   // logs '5'
console.log(numObj.toPrecision(2));   // logs '5.1'
console.log(numObj.toPrecision(5));   // logs '5.1235'
```

7.- Tipo Número

- Convertir cualquier variable en un número
  - La función Number() permite convertir cualquier variable en un número

```
Number(true); // devuelve 1
var num = false;
Number(num); // devuelve 0

Number('123')  // retorna el número 123
Number('123') === 123  // retorna true
```

8.- Tipo Fecha

- Las fechas en JavaScript se pueden visualizar como un número o un string.
- JavaScript nos provee de un tipo de dato llamado Date, con el que podemos trabajar fácilmente con fechas de forma nativa y práctica.

CONSTRUCTOR	DESCRIPCIÓN
new Date()	Obtiene la fecha del momento actual.
new Date(str)	Convierte el texto con formato YYYY/MM/DD HH:MM:SS a fecha.
new Date(num)	Convierte el número num, en formato Tiempo UNIX, a fecha UTC.
new Date(y, m, d, h, min, s, ms)	Crea una fecha UTC a partir de componentes numéricos*.

8.- Tipo Fecha

- Para transformar una fecha a string, debe utilizarse el método toString()
- Podemos utilizar estas cuatro formas para crear fechas en Javascript.

```
 var objFecha1 = new Date();
 var objFecha2 = new Date("May 15, 2021");
 var objFecha3 = new Date("May 15, 2021 20:21:22");
 var objFecha4 = new Date(2021, 4, 15);
 var objFecha5 = new Date(2021, 4, 15, 20, 21, 22);
// Devuelve la fecha actual, por ejemplo: "Mon May 17 13:52:40 UTC+0100 2021"
document.write( objFecha1 + "<br />" );
// Devuelve "Sat May 15 00:00:00 UTC+0100 2021"
document.write( objFecha2 + "<br />" );
// Devuelve "Sat May 15 20:21:22 UTC+0100 2021"
document.write( objFecha3 + "<br />" );
// Devuelve "Sat May 15 00:00:00 UTC+0100 2021"
document.write( objFecha4 + "<br />" );
// Devuelve "Sat May 15 20:21:22 UTC+0100 2021"
document.write( objFecha5 + "<br />" );
```

8.- Tipo Fecha

- Formatos tipo fecha
  - En JavaScript, existen 4 tipos de formatos de fecha de JavaScript:
    - Formato corto.
    - Formato largo.
    - Formato completo.
    - Formato ISO.

```
// Formato corto
var v = new Date("08/16/2021");
// Formato largo
var v = new Date("Ago 16 2021");
// Formato completo
var v = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (W. Europe Standard Time)");
// Formato ISO
var d = new Date("2015-03-25");
```

8.- Tipo Fecha

- Métodos tipo fecha

MÉTODO	QUÉ HACE
getDate()	Devuelve el día del mes. Número entre 1 y 31
getDay()	Devuelve el día de la semana. Entre 0 (domingo) y 6 (sábado)
getFullYear()	Devuelve el año con 4 dígitos
getHours()	Devuelve las horas.
getMilliseconds()	Devuelve los milisegundos entre 0 y 9999
getMinutes()	Devuelve los minutos. Entre 0 y 59
getMonth()	Devuelve el mes. Entre 0 (enero) y 11 (diciembre)
getSeconds()	Devuelve los segundos. Entre 0 y 59
getTime()	Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje

9.- Arrays

- El **objeto Array** de JavaScript es un objeto global que es usado en la construcción de arrays, que son objetos tipo lista de alto nivel.
- Los arrays son una serie de elementos, a los cuales pueden accederse utilizando un subíndice.
- Tanto la longitud como el tipo de los elementos de un array son variables.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Título de la WEB</title>
</head>
<body>
    <p id="aqui"></p>
    <script>
        var dias = ["Lunes","Martes","Miercoles","Jueves","Viernes"];
       document.getElementById("aqui").innerHTML = dias;

    </script>
</body>
</html>
```

9.- Arrays

- Operaciones con arrays
  - Conocer su longitud:
    - longitud = dias.length;
  - Ordenarlo alfabéticamente:
    - dias = dias.sort;
  - Añadir un nuevo elemento al final:
    - dias = dias.push("Sabado");
  - Extraer el último elemento del array:
    - ultimo = dias.pop();
    - El último elemento se eliminará del array. Extraer significa "retirar".
  - Eliminar el primer elemento:
    - dias.shift();
  - Eliminar un elemento concreto:
    - delete dias[2];

9.- Arrays

- Acceder a un elemento del Array mediante su índice

```
let frutas = ["Manzana", "Banana"]
let primero = frutas[0]
// Manzana

let ultimo = frutas[frutas.length - 1]
// Banana
```

- Recorrer un Array
  - Se puede recorrer mediante el recorrido clásico con un for o mediante en método forEach().
    - Recorrido clásico: mediante bucles for

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

    - Recorrido mediante bucles foreach

```
var dias = ["Lunes","Martes","Miercoles","Jueves","Viernes"];
dias.forEach(muestra)
```

      - Donde muestra es una función que será invocada por cada uno de los elementos del array días

9.- Arrays

```
<body>
    <h3>Recorrdio de un array</h3>
    <p id="aqui"></p>
    <p id="aqui2"></p>
    <script>
        var txt = "";
        var txt2 = "";
        var dias = ["Lunes","Martes","Miercoles","Jueves","Viernes"];
        dias.forEach(muestra);
        document.getElementById("aqui").innerHTML = txt;

        function muestra(valor, index, array){
            txt = txt + valor + " ";
        }


        for (i=0;i<dias.length;i++){
            txt2 = txt2 + dias[i] + " ";
        }
        document.getElementById("aqui2").innerHTML = txt;

    </script>
</body>
```

9.- Arrays

- Resultado del ejemplo anterior

9.- Arrays

- Encontrar el índice de un elemento del Array

```
let frutas = ["Manzana", "Banana","Fresa"]
frutas.push('Fresa')
// ["Manzana", "Banana", "Fresa"]

let pos = frutas.indexOf('Banana') // 1
```

- Eliminar un único elemento mediante su posición

```
let frutas = ["Manzana", "Banana","Fresa"]
var pos = 1;
let elementoEliminado = frutas.splice(pos, 1)
// ["Manzana", "Fresa"]
```

9.- Arrays

- Eliminar varios elementos a partir de una posición
  - Con .splice() también es posible extraer elemento de un array y guardarlos en un nuevo array.
  - **¡Ojo!** que al hacer esto estaríamos modificando el array de origen.

```
let vegetales = ['Repollo', 'Nabo', 'Rábano', 'Zanahoria']
console.log(vegetales)
// ["Repollo", "Nabo", "Rábano", "Zanahoria"]

let pos = 1, numElementos = 2

let elementosEliminados = vegetales.splice(pos, numElementos)
// ["Nabo", "Rábano"] ==> Lo que se ha guardado en "elementosEliminados"

console.log(vegetales)
// ["Repollo", "Zanahoria"] ==> Lo que actualmente tiene "vegetales"
```

9.- Arrays

- Copiar un array

```
let copiaArray = vegetales.slice();
// ["Repollo", "Zanahoria"]; ==> Copiado en "copiaArray"
```

9.- Arrays

Arrays Multidimensionales

- Los arrays multidimensionales consisten en tener en una posición de una array otro array de elementos.
- Se pueden tener arrays bidimensionales, tridimensio-nales, etc.
  - Lo más común en programación es tener tablas, también llamados arrays bidimensionales.

```
var dias = [["lunes","martes","miércoles","jueves","viernes"], ["sábado","do-
mingo"]];
console.log(dias[0],[2]); // mostrará por consola miércoles
console.log(dias[1],[1]); // mostrará por consola domingo
console.log(dias[1]); // mostrará por consola sábado, domingo
```

10.- Sentencias condicionales y bucles

Sentencias de control

- En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control".
- Una estructura de control, dirige el flujo de ejecución a través de una secuencia de instrucciones, basadas en decisiones simples y en otros factores.
- Una parte muy importante de una estructura de control es la "**condición**".
  - Cada condición es una expresión que se evalúa a true o false.

10.- Sentencias condicionales y bucles

Sentencia if

- Se ejecutará el bloque de código si la condición es verdadera.
- Sintaxis:

```
if (condición)
// entre paréntesis irá la condición que se evaluará a true o false.
{
   // instrucciones a ejecutar si se cumple la condición
```

```
if (miEdad >30)
{
      alert("Ya eres una persona adulta");
}
```

10.- Sentencias condicionales y bucles

Sentencia if-else

- En este tipo de construcción, podemos gestionar que haremos cuando se cum-pla y cuando no se cumpla una determinada condición.
- Sintaxis:

```
if (condición) // entre paréntesis irá la condición que se evaluará a true o false.
{
        // instrucciones a ejecutar si se cumple la condición
}
else
{
        // instrucciones a ejecutar si no se cumple la condición
}
```

10.- Sentencias condicionales y bucles

- Estructura if-else-if
  - Es combinación de las anteriores y se utiliza cuando la primera condición sea falsa y se requiere otra condición.
  - Sintaxis:

```
if (condición) // entre paréntesis irá la condición que se evaluará a true o false.
{
// instrucciones a ejecutar si se cumple la condición
}
else if (condición) // entre paréntesis irá la condición que se evaluará a true o false.
{
// instrucciones a ejecutar si se cumple la condición
}
else
{
// instrucciones a ejecutar si no se cumple la condición
}
```

10.- Sentencias condicionales y bucles

```
if (miEdad >30)
{
alert("Ya eres una persona adulta.");
}
else
{
alert("Eres una persona joven.");
}
```

```
if (miEdad >30)
{
alert("Ya eres una persona adulta.");
} else if (miEdad < 14)
alert("Eres un adolescente.");
else
{
alert("Eres una persona joven.");
}
```

Estructura switch

- Evalúa una expresión, igualando el valor de la expresión a una cláusula case y ejecuta las sentencias asociadas con dicho case.

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

10.- Sentencias condicionales y bucles

```
switch (expr) {
    case 'Naranjas':
      console.log('El kilogramo de naranjas cuesta $0.59.');
      break;
    case 'Manzanas':
      console.log('El kilogramo de manzanas cuesta $0.32.');
      break;
    case 'Platanos':
      console.log('El kilogramo de platanos cuesta $0.48.');
      break;
    case 'Cerezas':
      console.log('El kilogramo de cerezas cuesta $3.00.');
      break;
    case 'Mangos':
    case 'Papayas':
      console.log('El kilogramo de mangos y papayas cuesta $2.79.');
      break;
    default:
      console.log('Lo lamentamos, por el momento no disponemos de ' + expr + '.');
  }

  console.log("¿Hay algo más que te quisiera consultar?");
```

10.- Sentencias condicionales y bucles

BUCLES

- Los bucles son estructuras repetitivas, que se ejecutarán un número de veces fijado expresamente, o que dependerá de si se cumple una determinada condición.

Bucle for

- Este tipo de bucle te deja repetir un bloque de instrucciones un número limitado de veces.
- Sintaxis:

```
for (expresión inicial; condición; incremento)
{
// Instrucciones a ejecutar dentro del bucle.
}
```

10.- Sentencias condicionales y bucles

```
for (var i=1; i<=20; i++)
{
    text += "Número: " + i + " ";
}
```

- Existe una variable del bucle for que es el bucle for in. A continuación, se muestra un ejemplo sencillo:

```
var persona = {nombre:"David", apellidos:"Moreno", edad:25};
var texto = "";
var x;
for (x in persona)
{
    text += persona[x];
}
```

Bucle While

- Este tipo de bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición.
- Sólo se indica la condición que se tiene que cumplir para que se realice una iteración o repetición.
- Sintaxis:

```
while (condición)
{
// Instrucciones a ejecutar dentro del bucle.
}
```

```
var i=0;
while (i <=10){
    text += "Número " + i;
    i++;
}
```

Bucle do-while

- El tipo de bucle do...while es la última de las estructuras para implementar repeticiones de las que dispone JavaScript, y es una variación del bucle while() visto anteriormente.
- Se utiliza generalmente, cuando no sabemos el número de veces que se habrá de ejecutar el bucle.
- Es prácticamente igual que el bucle while(), con la diferencia, de que sabemos seguro que el bucle por lo menos se ejecutará una vez.
- Sintaxis:

```
do {
    // Instrucciones a ejecutar
}while (condición);
```

```
var a = 1;
do{
    alert("El valor de a es: "+a); //Mostrará esta alerta 2 veces
    a++;
}while (a<3);
```