**CLASE 01 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

TypeScript es un lenguaje de programación de código abierto, que permite crear aplicaciones web en JavaScript. No requiere ningún tipo de plugin, genera código en JavaScript que se ejecuta desde cualquier navegador, plataforma o sistema.

TypeScript -> **Transpilador** (Un compilador que traduce las instrucciones de un lenguaje a otro.

**Comandos útiles:**

tsc <Nombre\_del\_archivo>.ts Se utiliza para que se transpile un archivo de TypeScript.

tsc -w <Nombre\_del\_archivo>.ts Se utiliza para agregar una inspección sobre un archivo.

tsc –outFile <Main> <Arch01>.ts <Arch02>.ts Se utiliza para juntar varios archivos .ts en una sola salida.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Tipos de datos**:

**Tipado** **estático** (Fuertemente tipado) -> Se debe definir el tipo de dato, obligando a que no haya errores con los tipos de datos.

**Tipado** **dinámico** (Débilmente tipado) -> No se deben de o tiene por que especificar el tipo de dato) (PHP).

**Tipos en TypeScript:** Boolean – Number – String – Any – Void – Array – Null – Undefined – Tuple – Enum.

Primitivos:

**Boolean** (True / False) **var** esVerdad : **Boolean** = false;

**Number** (Valores numéricos, enteros, decimales, etc.) **var** numero : **Number** = 33.78;

**Null** (Cuando un objeto o una variable no accesible) **var** obj : Object | **Null** = null;

**Undefined** (Cuando un objeto o variable existe, pero no tiene valor)

**Any** (Puede ser cualquier tipo de objeto de JavaScript) **var** cosa : **Any** = “Rojo”; cosa = 3;

**Void** (Generalmente usado en funciones) **function** Avisar() : **Void** { console.log(“Hola”); }

**String** (Cadenas de caracteres y/o textos) **var** color : **String** = “rojo”; **color** = ‘azul’; **color** = `verde`;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Arrays:**

Array (Si no se especifican son ANY) **var** lista = [1, true, “Rojo”];

**var** lista : **Number**[] = [1,2,3]; **var** lista : **Array**<**number**> = [1,2,3];

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Enums:**

Enum (Solo almacenan números para identificar a las constantes.

Sin asignación de valores -> **enum** Color {Rojo, Verde, Azul}; var c : Color = Color.Verde; **// 2**;

Con asignación de valores -> **enum** Color {Rojo = 2; Verde = 5; Azul = 8} var c : Color = Color.Verde **// 5**;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**LET VS VAR** -> En TypeScript hay 2 maneras de declarar variables. VAR no tiene un ámbito de bloque (Es global), mientras que LET tiene ámbito dentro del bloque donde se la declara.

**var** foo : **number** = 123;

if (true) {var **foo** : **number** = 456;}

console.log(**foo**); **// 456.**

**let** foo : **number** = 123;

if (true) {let **foo** : **number** = 456;}

console.log(**foo**); **// 123.**

**Funciones (Tradicionales)**

SINTAXIS: function Identificador ([Args : tipo]) : TipoRetorno {[Return;]}

**P. Opcionales:**  (?) function Identificador (param **?** : tipo) : TipoRetorno {[Return;]}

**P. Predeterminados:** (= valor) function Identificador (param : tipo = **valor**) : TipoRetorno {[Return;]}

**P. REST:**  (…params) function Identificador ( **…params** : tipo[]) : TipoRetorno {[Return;]}

**Como variables:**

let saludar : Function = function() : string

{

return "Hola Mundo!!!";

}

console.log(saludar()); **//Hola Mundo!!!**

**function** Cuadrado(a:**number**) : number

{ return a \* a; }

let pot : Function = Cuadrado;

console.log(pot(2)); **//4**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Sobrecargas:** Este tipo de sobrecarga no tiene mucho sentido porque sería más simple poner un parámetro de tipo ***any***.

function a(x : string) : Void;

function a(x : boolean) : Void;

**function a(x : any) : Void; -> Correcto.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Funciones Fat Arrow (Funcion flecha)**

El nombre **‘Fat Arrow’** surge como oposición a las flechas finas ->

Son funciones cuyo propósito es: Omitir las palabras **function** y **return**. Implementar el **this** léxico.

* **Parámetros con Fat Arrow:**

() => { …. } // sin parámetros, lleva paréntesis.

x => { …. } // un parámetro, puede no llevar paréntesis.

(x,y) => { …. } // varios parámetros, lleva paréntesis.

* **Parámetros con funciones ‘normales’:**

**function**() { …. }

**function**(x,y) { …. }

**Implementar el cuerpo:** **x => {return x\*x}** Equivale a: **x => x\*x**

El bloque de instrucciones se comporta como un cuerpo de función normal. Con un cuerpo de expresión, la expresión siempre se devuelve implícitamente. Se puede omitir el return. Tener un cuerpo de bloque agregado a un cuerpo de expresión significa que la expresión retornada es un objeto literal. Se debe poner entre paréntesis.

**let** res = () => {(“nombre” : “Juan”, “edad” : 23)};

Hasta las funciones flecha, cada nueva función define su propio valor de ***this***

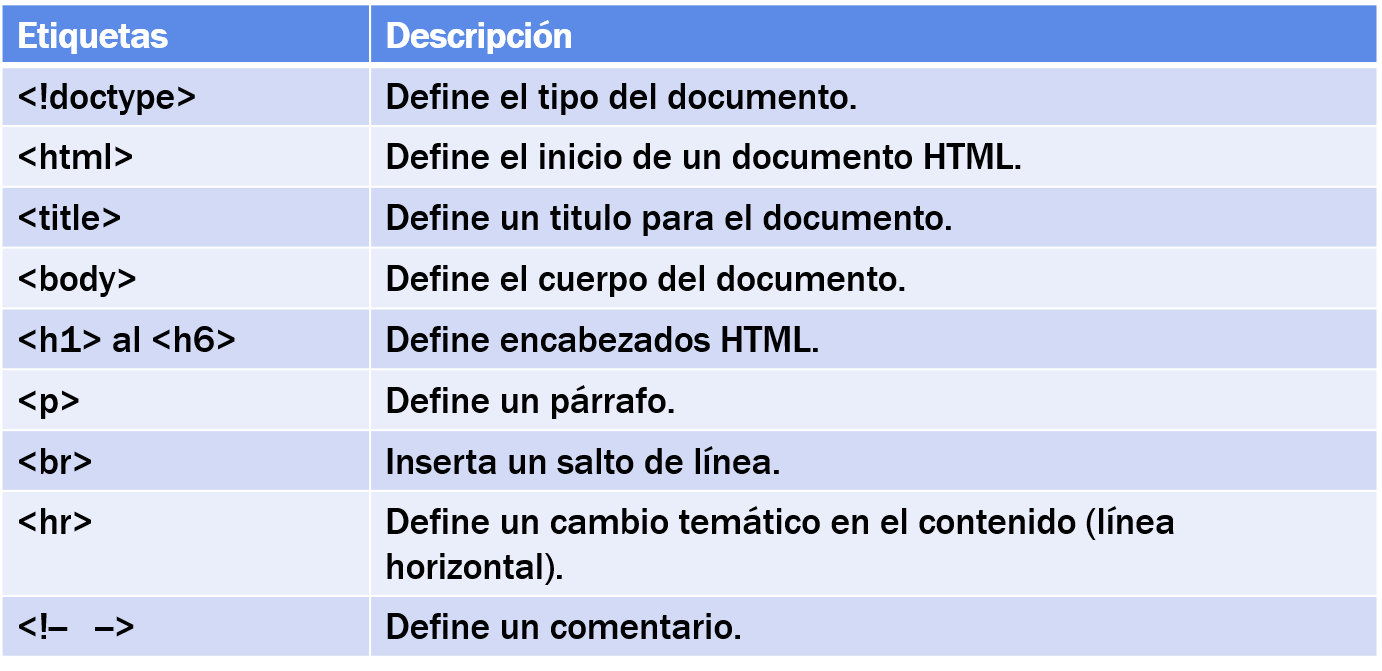
**CLASE 02 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**¿Qué es HTML / HTML 5?** Es un lenguaje utilizado para desarrollar paginas web. Utiliza una serie de tags intercaladas en un documento de texto sin formato. Dichas etiquetas son interpretadas por navegadores, y eso permite visualizar una página web.

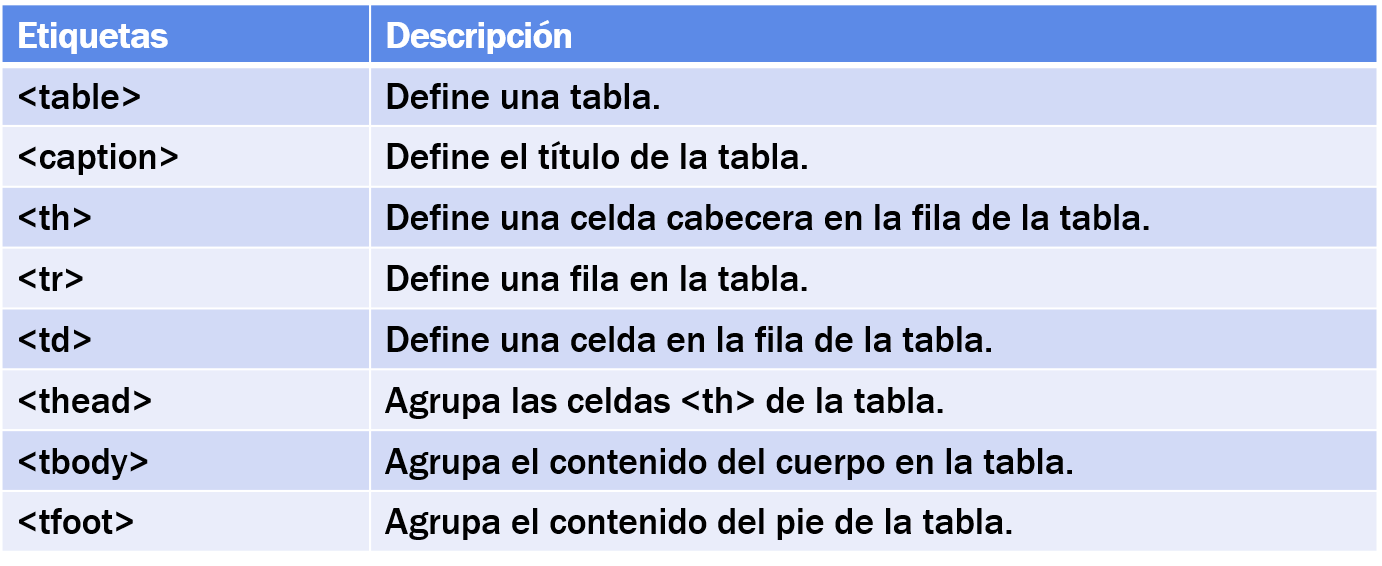
**Fusiona: JavaScript (Como lenguaje de programación) – HTML (Como modelo semántico) – CSS3 (Como lenguaje de estilos).**

La estructura de los tags es la siguiente: **<nombreEtiqueta atributo = “valor” > </nombreEtiqueta>**

**ELEMENTOS BASICOS:**



**TABLAS:**



**FORMULARIOS:** Nos permiten desde dentro de una aplicación Web, solicitar información al visitante.

**<form method=“[GET|POST]” action=“URL” name=“frm” id=“frm” >**

**Contenido del formulario**

**</form>**

**CONTROLES DE ENTRADA DE DATOS:**

**INPUT: Es un tag multifunción, de acuerdo al valor del atributo Type.**

<input type=“[TEXT|PASSWORD|CHECKBOX|RADIO|SUBMIT|RESET|FILE|HIDDEN|IMAGE|BUTTON]”

name=“nombre” id=“nombre” value=“valor” />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Caja de texto: Control de entrada de tipo *text*.**

<input type=“text” name=“txtDni” size=“35” value=“0” id=“txtDni” maxlength=“8” />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Casilla de verificación: Control de entrada *CheckBox*. Seleccionado / no seleccionado. Se debe especificar el name**

<input type=“checkbox” name=“chkOp1” value=“0” checked />

<input type=“checkbox” name=“chkOp2” value=“1” />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Boton de opción: Entrada de tipo *radio*. Seleccionado / no seleccionado. Se debe especificar el name.**

<input type=“radio” name=“rdoTipo” value=“dni” checked />

<input type=“radio” name=“rdoTipo” value=“Li” />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Parametros ocultos: Entrada de tipo *hidden*. No muestra ningún campo. No ocupan lugar en la página.**

<input type=“hidden” name=“variable” id=“variable” value=“valor” />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Enviar Datos: Es un control de entrada de tipo *submit*.** <input type=“submit” value=“Enviar Datos” />

**Borrar Datos: Es un control de entrada de tipo *reset****. <input type=“reset” value=“Borrar Datos” />*

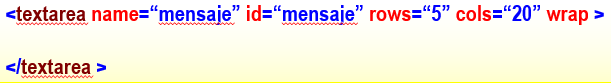
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Imágenes:** **Entrada de tipo *image*. Su finalidad es análoga al botón submit.**

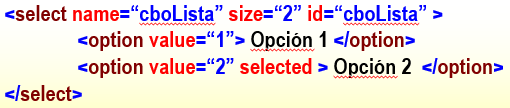
<input type=“image” src=“../Imagenes/miImagen.jpg” />

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

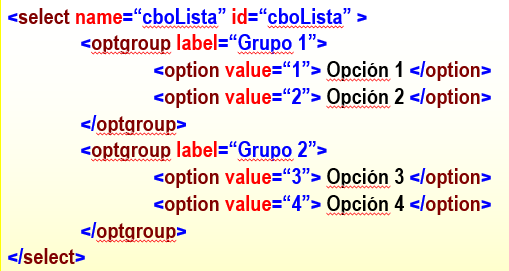
**CAJA DE TEXTO MULTIMEDIA:** El atributo ***rows*** especifica el número de filas que se visualizará en el área de texto. Mientras que el atributo ***cols*** indica la cantidad de caracteres por fila. El atributo ***wrap*** indica que se saltará automáticamente a la línea siguiente cuando se complete la línea en la que se escribe.



**LISTAS DESPLEGABLES:** La etiqueta que permite crear un control de este tipo es *<****select****>*. Las opciones se especifican con el tag *<****option****>*. El atributo ***value*** indica el valor asociado con la opción especificada. Si se omite, toma el valor del texto.El atributo ***size*** indica la cantidad de opciones que se visualizarán en la lista.

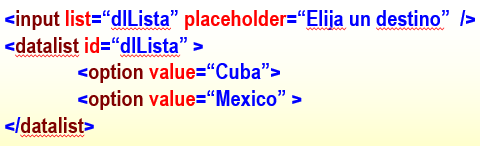


**OPCIONES AGRUPADAS:** La etiqueta utilizada es <**optgroup**>. Se utilizan como **submenúes**. Cada grupo quedará identificado con el atributo **label**.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

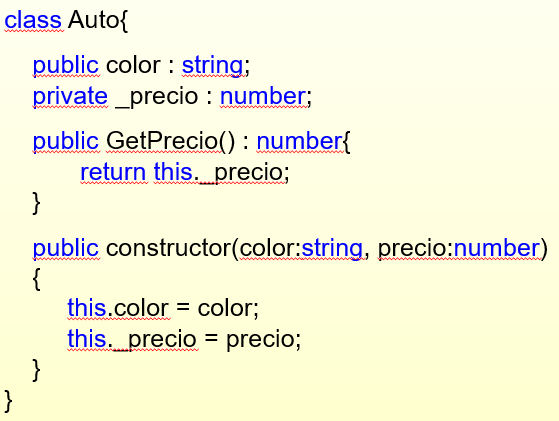
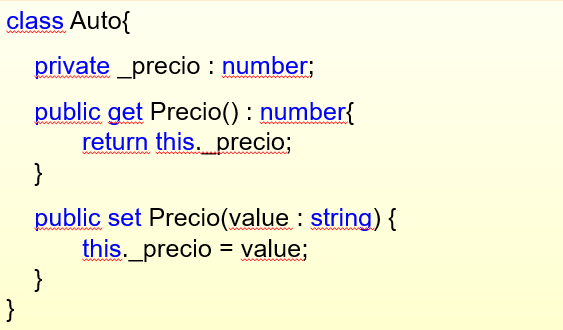
**Detalist (HTML 5)** La etiqueta <**datalist**> especifica una lista pre definida de opciones para un <**input**>. Las opciones se especifican con el tag <**option**>. El atributo **value** indica el valor asociado con la opción especificada. El atributo **list** del <**input**> debe referir al atributo id del <**datalist**>.



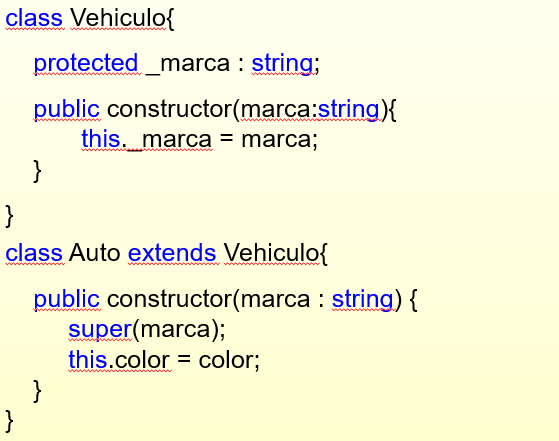
**CLASE 03 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**POO:** Los miembros de una clase poseen modificadores de visibilidad. Public (Defecto) – Protected – Private.

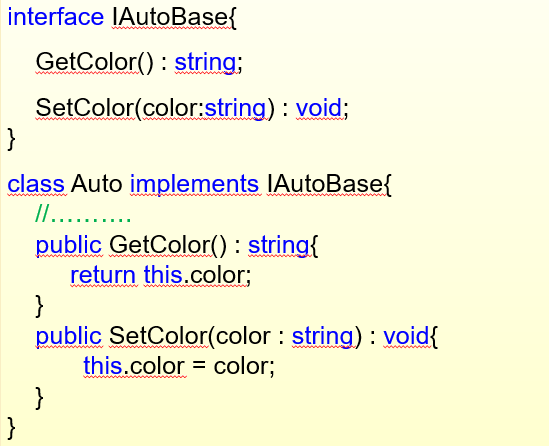
Los miembros de una clase pueden ser **estáticos** y no **estáticos**. Con el modificador **static**, se marcan los miembros estáticos de una clase.

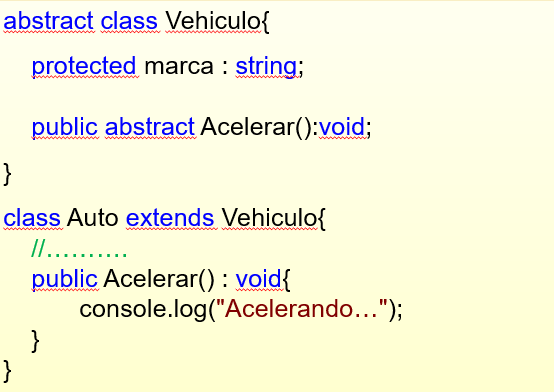
**HERENCIA:**



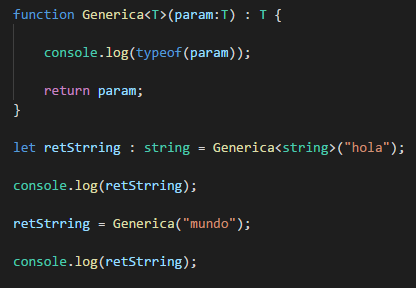
**INTERFACES:**

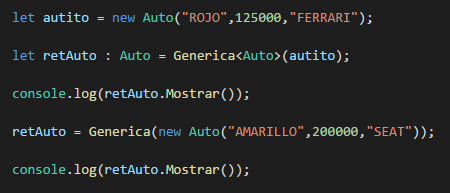


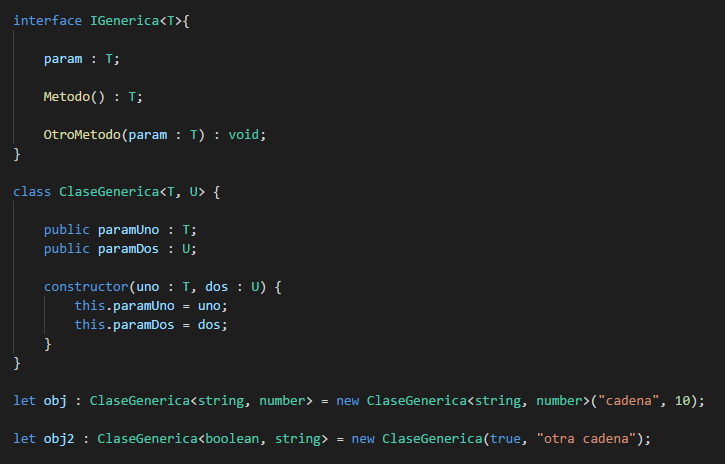
**CLASES ABSTRACTAS:**

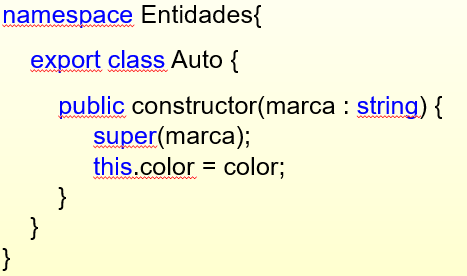


**GENERICS:**









**NAMESPACES:**

Concepto similar al que utiliza C#. Agrupaciones lógicas de elementos. Separación física de elementos de un mismo proyecto. Se pueden incluir clases, funciones, variables y otros namespaces. Para poder acceder a los miembros de un namespace (por fuera del archivo actual) se debe agregar la palabra reservada ***export***.

**CLASE 04 – AJAX \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**AJAX:** No es un lenguaje de programación, sino un conjunto de tecnologías que permiten hacer paginas de internet mas interactivas. Permite que las paginas sean capaces de **actualizar** partes en **segundo** **plano**, sin tener que recargarla.

**Tecnologías:** **XHTML y CSS:** Para crear una presentación basada en estándares. **DOM:** Para la interacción y manipulación dinámica de la presentación. **XML, XSLT y JSON:** Para el intercambio asincrónico de información (asincrónica es aquella comunicación que se establece entre personas de manera diferida en el tiempo) **JS:** Para unir todas las demás tecnologías.

**Ventajas:**

\* Utiliza tecnologías ya existentes.

\* Soportada por la mayoría de los navegadores modernos.

\* Interactividad: El usuario no tiene que esperar hasta que lleguen los datos del servidor

\* Portabilidad: No requiere plugin cómo Flash y Applet de Java)

\* Mayor velocidad, esto es debdo que no hay que actualizar la página de nuevo. \*La página asemeja una aplicación de escritorio.

**Desventajas:**

\*Problemas con navegadores antiguos, que no implementen esta tecnología.

\* No funciona si el usuario tiene desactivado JavaScript en su navegador.

\* Requiere programadores que conozcan todas las tecnologías que intervienen en AJAX.

**XMLHtpRequest**: La piedra angular de AJAX es este objeto. Es utilizado para **intercambiar** **datos** con el servidor en segundo plano.

Para enviar una petición al servidor, se utiliza los métodos **open() y send().**

**xhttp**.open("Método", "Url", Async (true / false));

* + Método: Especifica el tipo de pedido (GET/POST).
  + Url: Indica la ubicación del archivo en el servidor.
  + Async: true (Asincrónico); false (Sincrónico).

**xhttp**.send(); //Si es **GET**

**xhttp**.send("string"); //Si es **POST**

**Enviar una petición al servidor:**

var **xhttp** = new XMLHttpRequest();

**xhttp**.open("GET", "ajax\_test.php", true);

**xhttp**.send();

**onreadystatechange:** Este evento se dispara cada vez que cambia **readyState**. La propiedad **readyState** mantiene el estado de **XMLHttpRequest**. Existen 3 propiedades importantes del objeto **XMLHttpRequest**:

**\*\*\* onreadystatechange**: Almacena una función (o el nombre de una función) que se invoca de forma automática cada vez que cambia de propiedad readyState.

**\*\*\* readyState**: Mantiene el estado del XMLHttpRequest:

* + 0 : Solicitud no inicializado
  + 1 : Conexión de servidor establecido
  + 2 : Solicitud recibida
  + 3 : Solicitud de procesamiento
  + 4 : Solicitud terminada y respuesta lista

**\*\*\* status**: 200 –> OK; 404 –> No encontrado.

**CLASE 05 – JSON \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**JSON - ¿Qué es?:** Es un formato de intercambio de datos ligero. Es fácil para los seres humanos leerlo y escribirlo. Y es fácil para las maquinas analizarlo y generarlo. Su formato está basado en un conjunto de la sintaxis de JavaScript:

**\*Literales de Arrays, y literales de objetos.**

Es completamente independiente del lenguaje. Pero utiliza las convenciones que son familiares para los programadores de la familia de lenguaje C.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**¿En que se basa? Se basa en dos estructuras:**

\* Una colección de pares nombre / valor: En varios lenguajes, esto se realiza como un objeto, registro, estructura, diccionario, tabla hash, lista con clave o matriz asociativa.

\* Una lista ordenada de valores: En la mayoría de lenguajes, esto se realiza como una matriz, vector, lista o secuencia.

Estas dos estructuras de datos universales. Todos los lenguajes modernos los soportan de una forma u otra.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Literales de objetos:** Un objeto es un conjunto desordenado de pares nombre / valor. Un objeto comienza y termina con {}. Cada nombre es seguido por : y los pares nombre / valor están separados por , (coma).

var **persona** = **{** "nombre" **:** "Juan"**,** "edad" **:** 35 **}**;

**alert**(persona.nombre); // Juan

**alert**(persona.edad); // 35

**alert**(persona["nombre"]); // Juan

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Literales de Arrays:** Un array es una colección de valores. Comienzan y terminan con []. Los valores, separados por , (coma).

var **nombres** = **[** "Juan"**,** "Pablo"**,** "Ramiro" **]**;

**alert**(nombres[0]); // Juan

**alert**(nombres[2]); // Ramiro

var **personas** = **[{** "nombre" **:** "Juan"**,** "edad" **:** 35 **},**

**{** "nombre" **:** "Anibal"**,** "edad" **:** 26 **}]**;

alert(personas[0].nombre); // Juan

alert(personas[1].nombre); // Anibal

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SINTAXIS:** La sintaxis se JSON no es nada mas que la meszcla de literales de objetos y Arrays para almacenar datos. JSON representa solamente datos (No incluye el concepto de variables, asignaciones o igualdades).

//Este código…

var **personas** = **[{** "nombre" **:** "Juan"**,** "edad" **:** 35 **},{** "nombre" **:** "Anibal"**,** "edad" **:** 26 **}]**;

//quedaría…

**[{** "nombre" **:** "Juan"**,** "edad" **:** 35 **},**

**{** "nombre" **:** "Anibal"**,** "edad" **:** 26 **}]**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JSON.parse:** Este método analiza una cadena de texto como JSON, transformando opcionalmente el valor producido.

**var** str = **'**{ "nombre" : "Juan", "edad" : 35 } **'**;

**var** obj = JSON.parse(str); .

alert(obj.nombre); //Juan

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JSON.stringify** Este método convierte un valor a notación JSON representándolo.

\* Las propiedades de objetos **no-array** no se garantiza que se realicen en un orden particular.

\* Los objetos **Boolean**, **Number** y **String** se convierten a sus valores primitivos.

\* Si se encuentra **undefined** durante la conversión, o se **omite** (cuando se encuentra en un objeto) o se sustituye por **null** (cuando se encuentra en un array).

**var** obj = { "nombre" : "Juan", "edad" : 35 };

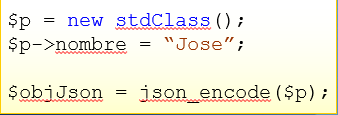
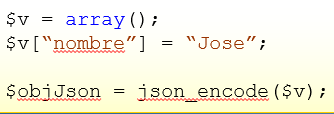
**var** str = **JSON**.**stringify**(obj);

// {"nombre" **:** "Juan"**,** "edad" **:** 35 }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JSON Y PHP:**

* Desde una aplicación en PHP podemos generar, a partir de objetos o arrays asociativos, objetos JSON para ser recuperados en JavaScript.
* Del mismo modo, podremos enviar desde JavaScript un objeto JSON que será recuperado en PHP como un array o un objeto.
* Las funciones ***json\_encode*** y ***json\_decode*** están diseñadas para tales propósitos.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CLASE 06 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***