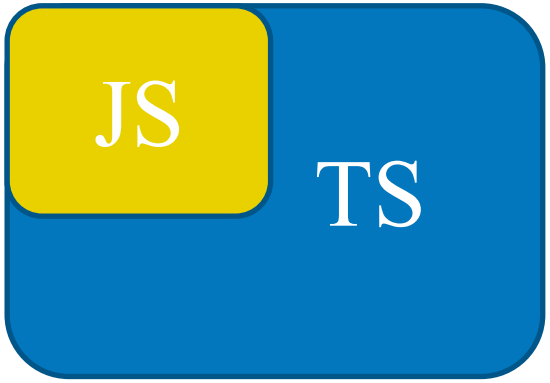
ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

TypeScripts
Fundamentos de la programación

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Para poder hablar de Typescript Primero hablaremos de JavaScript que tal vez estamos más familiarizados.

JavaScript es un lenguaje de programación que nos ayudará a definir la lógica de nuestro programa/ página web.
Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.



TypeScript se podría decir que es un super conjunto del JavaScript que ya se conoce, ya que es JavaScript pero con mas características y algunas características propias extendiendo así sus ventajas.
Es mantenido y actualizado por Microsoft.

¿Por qué JavaScript?

JavaScript es uno de los 3 lenguajes del lado del frontend que todos los desarrolladores web deben aprender dado que permite definir el comportamiento:

- **HTML** para definir el contenido de las páginas web
- **CSS** para especificar el diseño de las páginas web
- **JavaScript** para programar el comportamiento de las páginas web

JavaScript

Se trata de un **lenguaje de programación tipo script**, basado en objetos y guiado por eventos, diseñados específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet.

Los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

JavaScript es un **lenguaje de programación interpretado**, por lo que no es necesario compilar los programas para ejecutarlos.

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

¿Este código pertenece a TypeScripts o JavaScripts?

```
17  let nombre ='Juan';
18  let apellido = 'Perez'
19  let edad= 23;
20
```

¿Como puedo saber que pertenece a TypeScripts?

Para debatir en clase!!

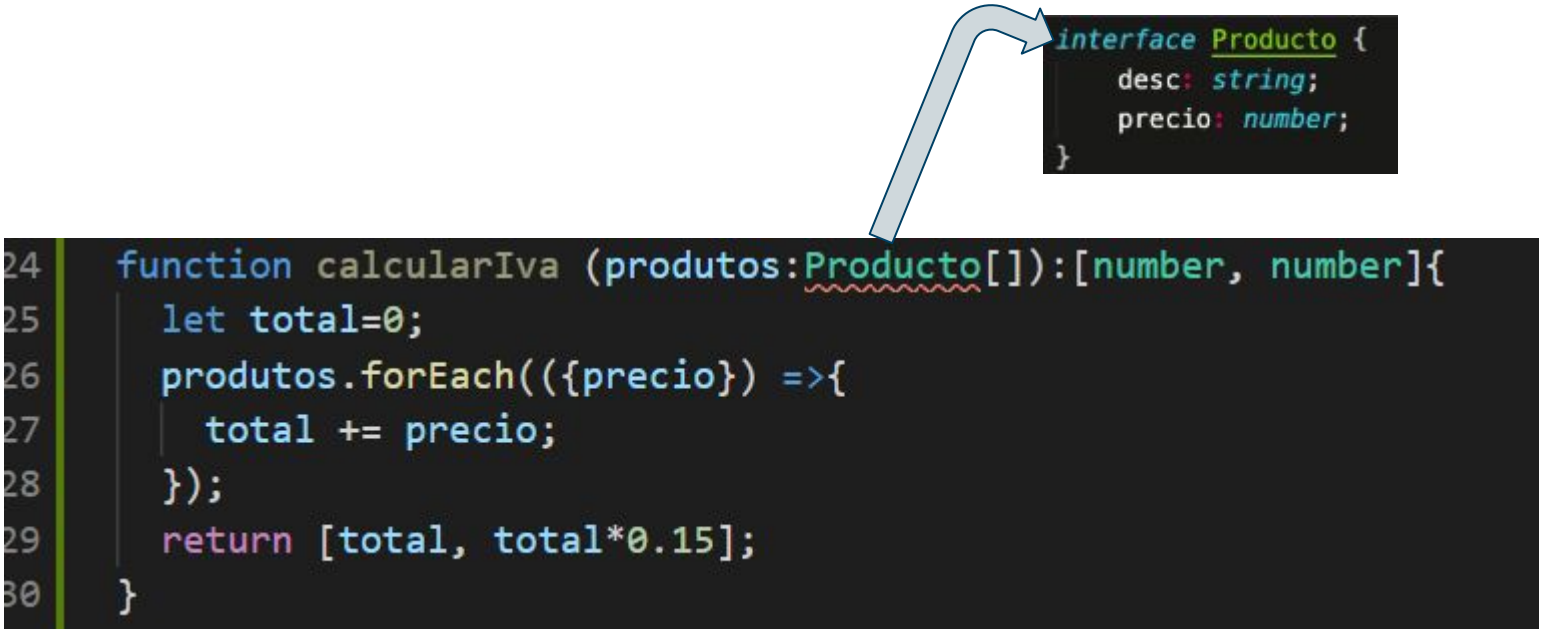
ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

```
20  function calcularIva (produtos){
21    let total=0;
22    produtos.forEach(({precio}) =>{
23      total += precio;
24    });
25    return [total, total*0.15];
26  }
27
```

Analiza y responde:

1. ¿Qué tipo de datos es producto?
2. ¿Qué propiedades debe tener producto?

Para debatir en clase!!



Typescript nos permite ver el tipo de variable en la misma linea de código.

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

¿Cual es el problema con JavaScript?

Problema – tipos dinámicos

- Las variables en JS son sin tipo y dinámicas lo que las hace flexibles pero, es muy fácil cometer errores.

- var x = 1; var y = x + 1;
  // OK, tipo es inferido. Podemos asumir que x e y ambas son numbers.

- var x = 1; x = "hello";
  // NOT OK, tipo es mezclado. Ahora no podemos asumir de qué tipo es la variable x.

- var int X = "hello";
  // NOT OK.

  Nota: JS es interpretado. No hay design-time o compile-time assistance para ayudar a encontrar errores

Problema - parámetros

- Definir una función con argumentos. Esto es un contrato con la llamada de la función. Desafortunadamente, en JS, los parámetros de funciones son más parecidos a guías y no un contrato en sí.

- function f(x) { return x + 1; }
- f("hello");
  f(1234);
  f();
  f(function(){});
  f([4]);
  f("hello", "world");

- // and then we have this magic object.
  function f() { console.log(arguments.length); }
  f(1,1,2,2);
  // De donde salió *arguments*?

Por lo que puedes observar, los programadores pueden cometer estos errores todo el tiempo. Deben tomar una actitud ordenada y controlar absolutamente todo.

// Es muy fácil cometer errores

Problema – extensión de objetos, no basados en clases

- JS es basado en objetos, no orientado a objetos. Permite manipular el DOM (Document Object Model)
- Esto puede ser bueno dado que no está restringido a la definición de sus objetos pero, podemos cometer errores fácilmente.

- ```
  var x = { a : 1, b : 2 }
  x.c = 3;
  console.log(x.a + x.b + x.c);
  ```

- ```
  var x = { a : 1, b : 2, a : 3 }
  console.log(x.a + x.b);
  ```

Nota: La herencia de objetos es posible, pero muy compleja en javascript. Esto significa que no tenemos un contrato definido en nuestro código, no describimos la forma y el comportamiento de nuestros objetos.

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

DEMO (Javascript)

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

¿Por que angular utiliza TypeScripts?

+

Por el beneficio que proveen los decoradores, para poder identificar las clases y el tipado.

Introducción a TypeScript - ¿Qué es TS?

TypeScript inicia desde la misma sintaxis y semántica de JavaScript que millones de desarrolladores conocen:

```typescript
// TypeScript
function unaFuncion( mensaje : string) {
    console.log("El mensaje es: " + mensaje);
}
```

```javascript
// JavaScript
function unaFuncion(mensaje) {
    console.log("El mensaje es: " + mensaje);
}
```

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

Sintaxis

TypeScript – Variables

Una variable es un espacio de memoria que se utiliza para almacenar un valor/dato durante un tiempo específico en que se ejecute el programa. Tiene un identificador (nombre) y un tipo de datos.

Ejemplos de variables nativas de TypeScript:

```typescript
// TypeScript
//String
let color: string = "blue";
color = 'red';
let fullName: string = `Bob Bobbington`;
let age: number = 37;
// Arrays:
let list: Array<number> = [1, 2, 3];
```

```javascript
// JavaScript
//String
var color = "blue";
color = 'red';
var fullName = "Bob Bobbington";
var age = 37;
// Arrays:
var list = [1, 2, 3];
```

Declaración de Variables

Al igual que javascript, las variables pueden ser declaradas como sigue :

var

Es el tipo de declaración más común utilizada. No tiene ámbito de bloque.

```
var medida= 10;
var m=10;
```

let

Es un tipo de variable mas nuevo, permite reducir algunos problemas que presentaba la sentencia var dado que el ámbito de la variable está definido en el bloque donde se declara.

```
let precio=0;
let b="hola mundo";
```

Tipos de Datos

string:

Representa valores de cadena de caracteres (letras); ej: “Hola, Mundo”

```
let saludo: string="hola, mundo";
let colorProducto = 'blue';
```

number:

Representa valores numéricos, como enteros (int) o decimales (float), también se utiliza el valor de (number); ej: int = 25.63.

```
let decimal : number codigoProducto = 6;
```

Tipos de Datos

boolean

Es un tipo de variable que puede tener solo dos valores, Verdadero (true) o Falso (false); ej: bandera = true .

```
let nombreProducto: boolean = false;
```

arrays

Este tipo de variables es una colección o grupo de datos, pueden ser

```
let lista : number[]=[1,2,3];
let list : string[]=['pimiento','papas','tomate'];
```

Tipos de Datos

any

Puede ser de cualquier tipo y su uso esta justificado cuando no tenemos información a priori del tipo de dato.

```
let cantidad: any= 4;
let desc: any [] =[1,true,"verde"]
```

void

Es un tipo muy común para un tipo retorno de función que no devuelve ningún valor.

```
function usuario(): void {
  console.log("Este es un mensaje para el usuario" );
}
```

Tipos de Datos

undefined

Tipo de dato que se suele trabajar con undefined principalmente cuando declaramos que una variable trabajará con el tipo de dato any.

```
let nodefinidio: undefined= undefined;
```

null

Tipo de dato vacío (definido como vacío) que se suele trabajar con undefined principalmente cuando declaramos que una variable trabajará con el tipo de dato any

```
let vacio: null= null;
```

Tipos de Datos

objeto

Es un tipo un tipo de dato que engloba a la mayoría de los no primitivos

```
let persona:object={nombre:"Ana", edad:45}
```

Array y Tuplas

Array de que contiene textos

```
let planetas: string[]= ['Mercurio', 'Venus', 'Tierra'];
```

Array de que contiene números

```
let masas: number[]= [13,28,15];
```

Array con booleanos

```
let rocosos:  boolean[] = [true, false, false, true]
```

Array que contiene cualquier elemento

```
let perdidos: any[] = [9, true, 'asteroides']
```

Tupla, los elementos son limitados y de tipos fijos

```
let diametro: [string, number] = ['Saturno', 116460]
```

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

Demo (Typescript)

Declaración de variables y asignación

Desestructuración

La desestructuración nos permite obtener valores de un array u objeto.

Ejemplo de desestructuración de objetos::

```
var obj={x:1,y:2,z:3};
console.log(obj.y);
```

Ejemplo de desestructuración de arrays:

```
var array=[1,2,3];
console.log(array[2]);
```

Ejemplo de desestructuración de arrays con estructuración:

```
var array_2=[1,2,3,5];
var [x,y, ...rest]= array_2;
console.log(rest);
```

Estructuración

La estructuración permite facilita que una función reciba una gran cantidad de parámetros como array.

```
function rest(first, second, ...allOthers)
{console.log(allOthers);}
```

Luego, llamando a la función y pasando muchos parámetros:

```
rest('1', '2','3','4','5');
```

La función imprimirá en consola, un array con los valores restantes (allothers)

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

¿Y qué con las fechas?

Typescript nos provee un objeto Date para manipular las fechas y el tiempo. Este objeto nos permite obtener o configurar el mes, el día, el año, horas, minutos y milisegundos.

¿Cómo crear un objeto Date?

```
let date: Date = new Date();
console.log("Date = " + date);

date= new Date("2021-06-20");
console.log("Date = " + date);

date = new Date(2018, 5, 5, 17, 23, 42, 11);
console.log("Date = " + date);
```

Métodos que provee el objeto Date

Método	Descripción
Date()	Retorna la fecha/tiempo
getDate()	Retorna el día
getMonth()	Retorna el mes
setDate()	Es usado para configurar el día
setMonth()	Es usado para configurar el mes
Entre otros	

Para más información clic aquí

Métodos que provee el objeto Date

Ejemplo

```
let date: Date = new Date();
date.setDate(13);
date.setMonth(11);
date.setFullYear(2021);

console.log("Year = " + date.getFullYear());
console.log("Date = " + date.getDate());
console.log("Month = " + date.getMonth());
console.log("Day = " + date.getDay());
```

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Actividad: Haciendo uso de typescript intenta escribir el fuente para calcular la edad de una persona.

Para debatir en clase!!

Operadores Matemáticos

Operadores matemáticos

Símbolo	Operador	Descripción
+	Suma	Suma dos números
-	Resta	Resta dos números
*	Multiplicación	Multiplica dos números
/	División	Divide dos números
%	IModulo	Devuelve el resto de dividir de dos números
++	Incremento	Suma 1 valor al contenido de una variable
--	Decremento	Resta un valor al contenido de una variable

https://www.w3schools.com/js/js_operators.asp

Operadores de Asignación

Operator	Example	Same As
=	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

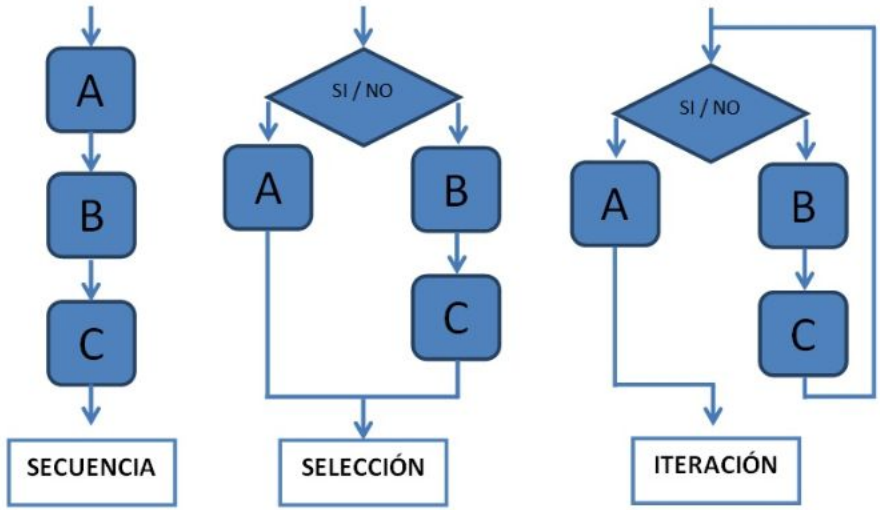
https://www.w3schools.com/js/js_operators.asp

Estructuras de Control

Son aquellas estructuras que permiten modificar el flujo de ejecución de las instrucciones de un programa.

Existen 3 estructuras de control que hacen que un programa o algoritmo sea fácil de entender, depurar o cambiar.

- **Secuencia**. Ejecuta las acciones sucesivamente una a continuación de otra.
- **Decisión (selección)**. Ejecuta una determinada secuencia de instrucciones dependiendo del valor de una determinada condición
- **Repetición (iteración)**. Secuencia de instrucciones que se ejecutan un determinado número de veces.



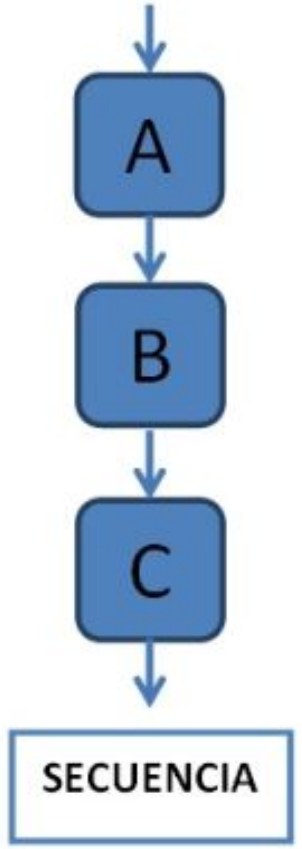
Estructuras de Control Secuencial

El término estructura de control se refiere al orden en que se ejecutan las instrucciones. Otros términos utilizados son: secuenciación y control de flujo.

A menos que se especifique, el flujo normal de control de todos es secuencial. Es decir que, las instrucciones se ejecutan una tras otra, en el orden en que se sitúan en el algoritmo.

Sin embargo, esta ejecución secuencial, no siempre puede resolver el problema por lo que, muchas veces necesitamos alterar o modificar el flujo de control.

Por ello, se definen otras estructuras de control tales como la estructura de control condicional y la de iteración.

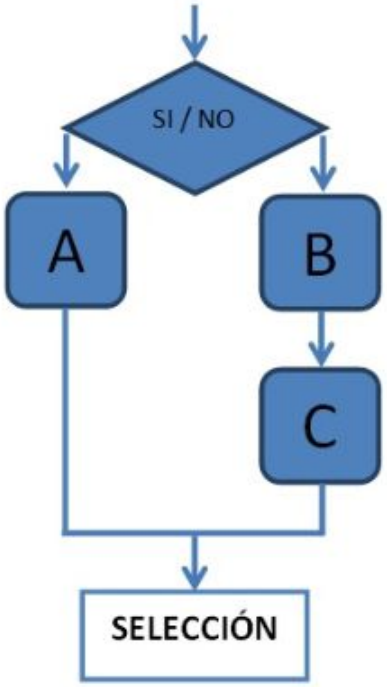


Estructura de Control Condicional

Se utiliza para indicarle a la computadora que debe evaluar una condición y, a partir del resultado, ejecutar el bloque de instrucciones correspondiente.

Se clasifican en:

- Simple
- Doble
- Múltiple

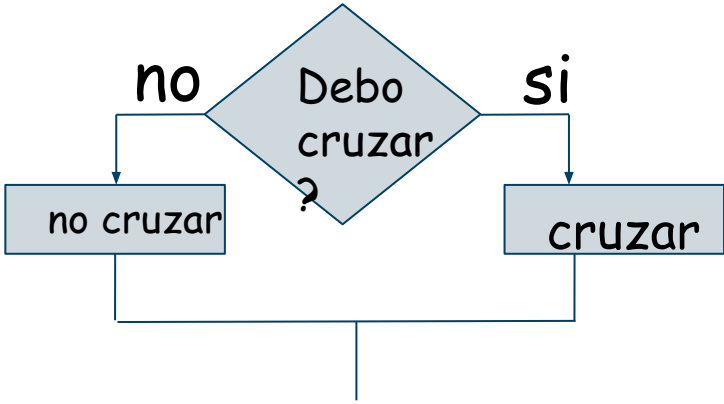


if… else

Ejecuta un determinado bloque de acciones cuando se cumple la condición.

Sintaxis:

```
 if...else:
if (condición) {
    sentencia_1;
 } else {
    sentencia_2;
}
```



Ejemplo:

```
if (hora < 18) {
  saludo = “Buen dia";
} else {
  saludo = “Buenas noches";
}
```

Operadores lógicos

Operador	Descripción	Ejemplo
Igualdad (==)	Devuelve **Verdadero (true)** si ambos operandos son iguales.	3 == var1 "3" == var1
Desigualdad(!=)	Devuelve **Verdadero (true)** si ambos operandos no son iguales.	var1 != 4 var2 != "3"
Mayor que (>)	Devuelve **Verdadero (true)** si el operando de la izquierda es mayor que el operando de la derecha.	var2 > var1 "12" > var1
Mayor o igual que (>=)	Devuelve **Verdadero (true)** si el operando de la izquierda es mayor o igual que el operando de la derecha.	var2 >= var1 var1 >= 3
Menor que (<)	Devuelve **Verdadero (true)** si el operando de la izquierda es menor que el operando de la derecha.	var1 < var2 "2" < 12
Menor o igual que (<=)	Devuelve **Verdadero (true)** si el operando de la izquierda es menor o igual que el operando de la derecha.	var1 <= var2 var2 <= 5

Operador condicional (ternario)

El operador condicional es el único operador que necesita tres operandos. El operador asigna uno de dos valores basado en la condición especificada.

Sintaxis:

```
condición ? valor1 : valor2
```

Si el resultado de la condición es true(verdadero), el operador tomará el primer valor, de lo contrario tomará el segundo valor especificado.

Ejemplo:

```
var estado = (edad >= 18) ? "adulto" : "menor";
```

Esta sentencia asigna el valor adulto a la variable estado si edad es mayor o igual a 18, de lo contrario le asigna el valor menor.

Operador coma

El operador coma (,) simplemente evalúa ambos operandos y retorna el valor del último. Este operador es por lo general utilizado dentro de un ciclo for, permitiendo que diferentes variables sean actualizadas en cada iteración del ciclo.

Por ejemplo, si a es un Array bi-dimensional con 10 elementos en cada lado, el siguiente código usa el operador coma para actualizar dos variables al mismo tiempo.

Ejemplo:

```
for (var i = 0, j = 9; i < 9; i++, j--)
  console.log("a[" + i + "][" + j + "]= " + a[i][j]);
```

El código imprime en la consola los valores correspondientes a la diagonal del Array

Switch

Una sentencia **switch** permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable.

```
switch (expresión) {
 case etiqueta_1:
      sentencias_1
      [break;]
  case etiqueta_2:
      sentencias_2
      [break;]
   ...
  default:
      sentencias_por_defecto
      [break;]
}
```

La estructura **Switch** evalúa el contenido de una expresión y:

- ejecuta la secuencia de instrucciones asociada al valor obtenido esa expresión.
- Opcionalmente, se puede agregar una opción final, denominada “default:”, cuya secuencia de instrucciones asociada se ejecutará sólo si el valor obtenido de la expresión no coincide con ninguna de las opciones anteriores.

Switch

Ejemplo:

```
var x = "0";
    switch (x) {
      case "0":
       console.log("Off");
        break;
      case "1":
        console.log("On")
        break;
      default:
        console.log("Default")
}
```

Estructuras Repetitivas

Las **estructuras repetitivas o cíclicas** nos permiten ejecutar varias veces un conjunto de instrucciones. A estas repeticiones se las conoce con el nombre de ciclos o bucles.

Estructuras repetitivas en JavaScript:

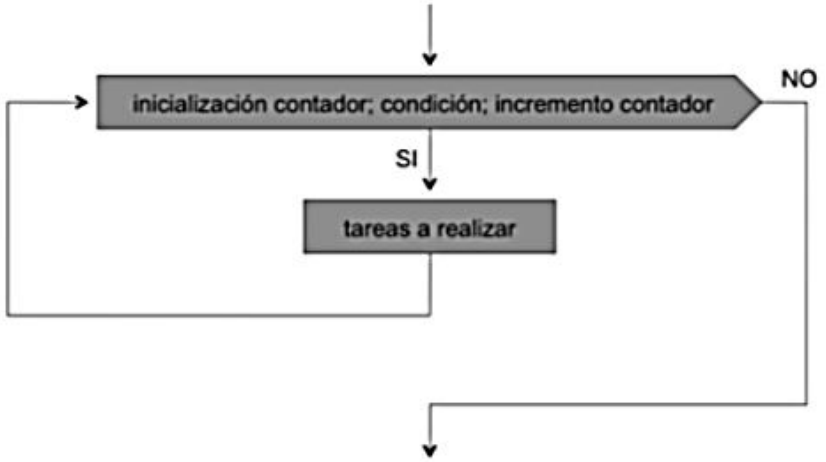
- FOR
- WHILE
- DO WHILE

For

En un bucle for se repite un bloque de código hasta que la condición especificada se evalúa como false. Es decir, se ejecuta un número determinado de veces. Este número de repeticiones es sabido de antemano.

Sintaxis:

```
 for ([expresionInicial]; [condicion]; [expresionIncremento])
{
...
}
```



For in

El bucle for in permite recorrer objetos y arrays.

Ejemplo:

```
let list = {a: 1, b: 2, c:3};
for(let i in list)
{console.log(list[i]); // 1, 2, 3}
```

while

El **bucle while** o **bucle mientras** es un ciclo repetitivo basado en los resultados de una expresión lógica. El propósito es repetir un bloque de código mientras la expresión lógica es verdadera.

Sintaxis:

```
while ([expresionInicial])
{...}
```

Ejemplo:

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

do while

El **bucle do while** o **bucle hasta** es un ciclo repetitivo basado en los resultados de una expresión lógica. El propósito es repetir un bloque de código hasta que la expresión lógica sea verdadera.

Sintaxis:

```
do {
 ...
} while ([expresión lógica]);
```

Ejemplo:

```
do {
  i = i + 1;
  result = result + i;
} while (i < 5);
```

break

La instrucción break para salir del bucle. Es decir que, rompe el ciclo y continúa ejecutando el código después del ciclo (si lo hay):

Sintaxis:

```
break;
```

Ejemplo:

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

Para más información consulta: https://www.w3schools.com/js/js_break.asp

continue

La sentencia continue puede usarse para reiniciar una sentencia while, do-while, for, o label.

La instrucción continue rompe una iteración (en el ciclo), si ocurre una condición especificada, y continúa con la siguiente iteración en el ciclo.

Sintaxis:

```
continue;
```

Ejemplo:

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

¿Qué es una función?

- Es un **conjunto de instrucciones** o sentencias que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.
- Son invocadas por su nombre.
- Permiten **simplificar el código** haciendo más legible y reutilizable.
- Puedes crear tus propias funciones y usarlas cuando sea necesario.

Declaración de Funciones

La declaración de una función consiste en:

- Un nombre
- Una lista de parámetros o argumentos encerrados entre paréntesis.
- Conjunto de sentencias encerrada entre llaves.

Sintaxis:

```
function nombre (parámetro1, parámetro2)
{
codigo ha ser ejecutado;
}
```

Ejemplos:

```
function calcularIva (produtos:Producto[]):[number,
number]{
  let total=0;
  produtos.forEach(({precio}) =>{
    total += precio;
  });
  return [total, total*0.15];
}
```

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Actividad:

Haciendo uso de typescript intenta escribir una función:

- que muestre por consola los números pares del 2 al 20.
- que calcule el factorial de cualquier número.

Para debatir en clase!!

Juego del número secreto.

El usuario irá introduciendo números por teclado, y el ordenador le irá dando pistas por consola: "el número es mayor" o "el número es menor", hasta que el usuario acierte. Cuando el usuario acierte, se le felicitará y se le comunicará el número de intentos que necesitó para acertar el número secreto por consola.

Para trabajar en clase!!

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

¿Qué es el DOM?

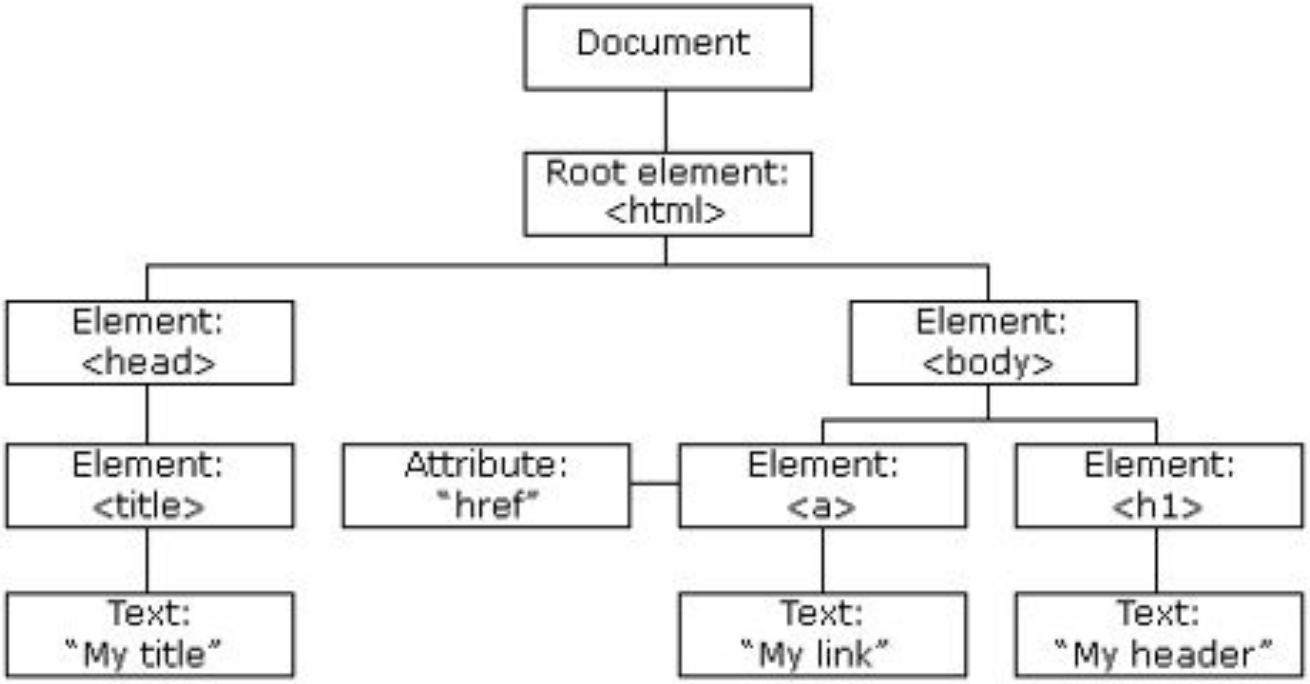
Para debatir en clase!!

¿Qué es el DOM?

- Es un **modelo de objetos de documento** (DOM) del W3C es una interfaz de lenguaje-neutral que define la estructura lógica de los documentos web y el modo en que se accede y manipula.
- Con dicho modelo los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar, o eliminar elementos y contenido.

Puedes consultar la definición del dom en
https://www.w3.org/2005/03/DOM3Core-es/introduccion.html

Cuando se carga una página, el navegador crea una jerarquía de objetos en memoria que sirven para controlar los distintos elementos de dicha página.



Fuente de la imagen:
https://www.acontracorrientech.com/wp-content/uploads/2019/09/blogPostPic-6.png

DOM

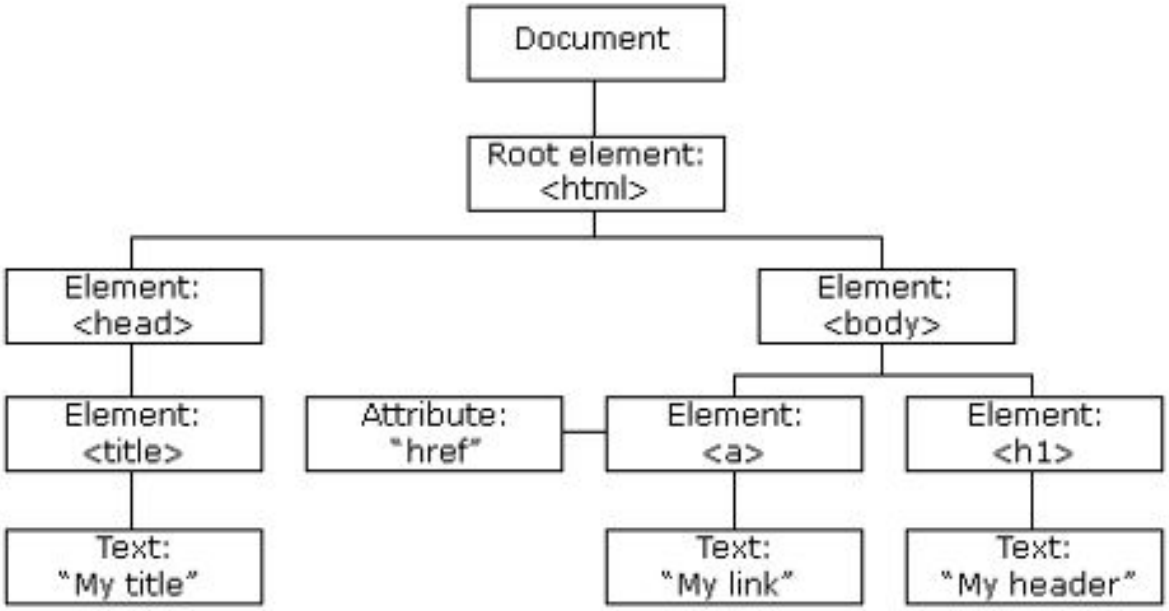
Define:

- Los elementos HTML como **objetos**.
- Las **propiedades** de todos los elementos HTML.
- Los **métodos** para acceder a todos los elementos.
- Los **eventos** para todos los elementos.

DOM es la forma de obtener, cambiar, agregar o borrar elementos HTML

DOM

- **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
- **Element**, representa cada una de los elementos HTML. Se trata del único nodo que puede contener atributos y e único del que pueden derivar otros nodos.
- **Attr**, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.
- **Text**, nodo que contiene el texto encerrado por una etiqueta HTML.



Fuente de la imagen: https://www.acontracorrientech.com/wp-content/uploads/2019/09/blogPostPic-6.png

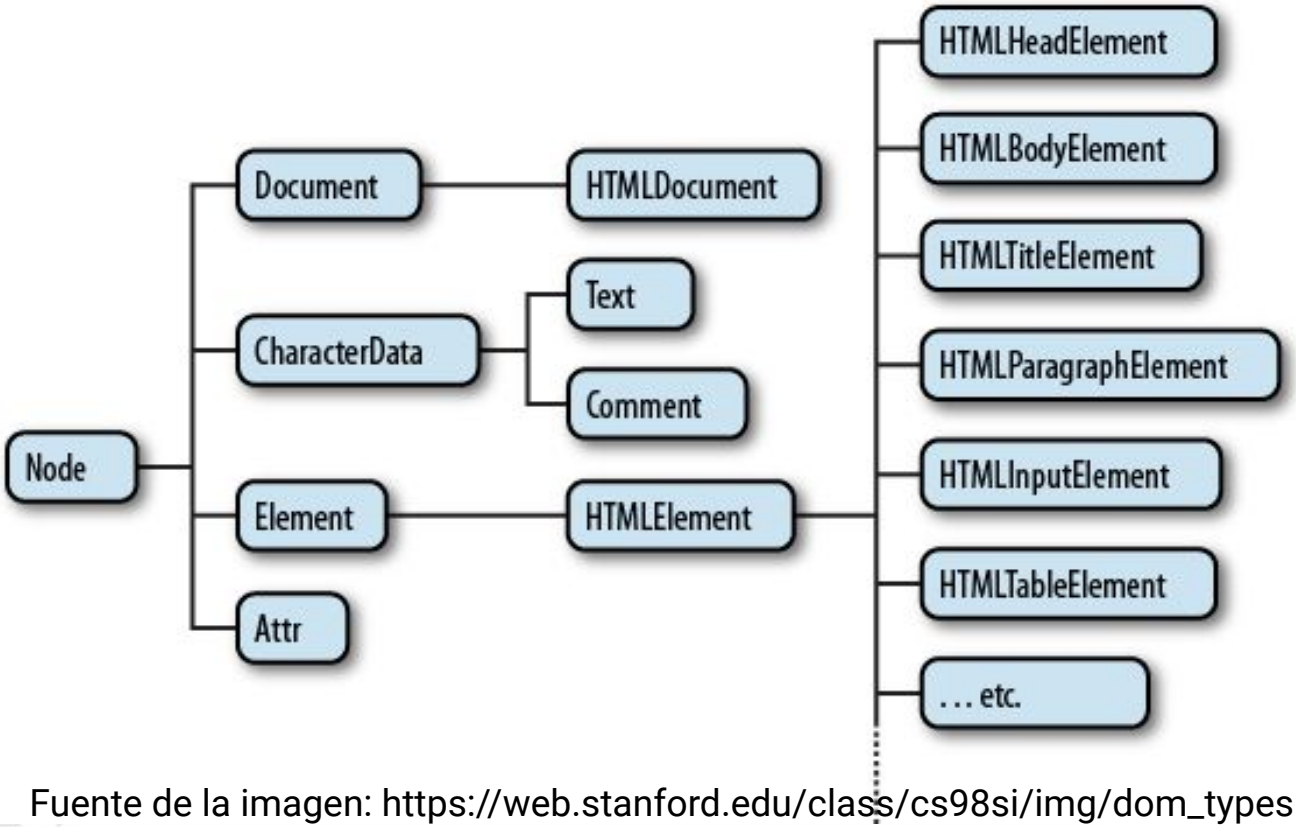
Manipulación del DOM mediante Typescript

Mediante **Typescript** podemos:

- Cambiar todos los elementos HTML de la página web.
- Cambiar los atributos de los elementos HTML.
- Cambiar los estilos CSS de los elementos HTML.
- Eliminar elementos HTML y atributos.
- Agregar nuevos elementos HTML y atributos.
- Crear nuevos eventos HTML.

Manipulación del DOM mediante Typescript

Al ser typescript fuertemente tipado, debemos identificar los objetos que representan cada elemento (aunque si no lo haces, typescript lo hará por tí)



Fuente de la imagen: https://web.stanford.edu/class/cs98si/img/dom_types.png

Manipulación del DOM mediante Typescript

- **body: HTMLElement**
  - hace referencia al elemento body.
- **getElementById(id: string): HTMLElement | null**
  - obtiene el elemento mediante el atributo 'id'
  - retorna un HTMLElement o null en caso de que no exista.
- **createElement(tag: string): HTMLElement**
  - crea un nuevo elemento
- **appendChild(child: HTMLElement): void**
  - agrega un elemento al documento

Podemos crear variables para acceder a métodos y propiedades del objeto documento con el objeto de acceder y modificar objetos DOM con TypeScript.

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Ejemplo sobre cómo insertar un elemento h1 de manera dinámica con Typescript:

```
let body: HTMLElement = document.body;
body.style.background = "lightblue";
let heading: HTMLHeadingElement = document.createElement("h1");
heading.style.color = "darkblue";
heading.innerText = "Manipulando el DOM desde Typescript";
body.appendChild(heading);
```

Type Assertion

Cuando usamos el método **getElementById**, este retornará un **HTMLElement** or null por lo que será necesario especificarle a typescript el tipo como sigue:

```
let h2:HTMLHeadingElement= document.getElementById("subtitulo") as
HTMLHeadingElement;
console.log(h2.innerText);
```

Esto permitirá el acceso a los métodos y propiedades que existen de ese tipo en específico y que no están definidas en el **HTMLElement**.

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

¿Cómo manipular el DOM desde Angular?

Recomendado

Renderer2

En Angular, la manipulación directa del DOM puede crear un acoplamiento indeseado entre la capa de renderizado y la de lógica. Sobre todo en los web workers y desktops.

Para sortear problema Angular proporciona una API llamada **Renderer2** para acceder de forma segura a elementos nativos.

Renderer2

Para ello, lo primero que debemos hacer es importar Renderer2 e inyectarlo como sigue:

```
import { Component, OnInit, Renderer2 ,ElementRef, ViewChild } from '@angular/core';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {
  @ViewChild('myheader') myHeader: ElementRef;

  constructor(private myRender2: Renderer2) {
   }
  ngOnInit(): void {
  }

}
```

myheader, debe estar referenciado en la vista como #myheader

Para comprender la definición del @ViewChild clic aquí

Modificar las clases

Podemos añadir o eliminar clases como sigue:

```
ngAfterViewInit():void{
  this.myRender2.removeClass(this.myHeader.nativeElement, "bg-dark");
  this.myRender2.addClass(this.myHeader.nativeElement,"bg-success");
}
```

Nota: en el ejemplo, estamos añadiendo y eliminando la clase después que la vista inicia. No podemos hacer uso del *this.myRender2* en el método onInit del componente puesto que en ese momento aún no se ha iniciado el render.

Añadir o eliminar atributos

Para añadir atributos:

```
this.myRender2.setAttribute(this.myHeader.nativeElement,
"title", "Este es el header");
```

Para eliminar atributos:

```
 this.myRender2.removeAttribute(this.myHeader.nativeElement,
"title");
```

Llamar a métodos de un elemento

Para acceder a la documentación de Renderer2 de angular clic aquí.

```
this.myRender2.selectRootElement(this.myButton.nativeElement).click();
```

Para más información hacer clic aquí.

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Material Complementario

https://tutorialesenpdf.com/typescript/previsualizacion/Manual-TypeScript.pdf
https://www.typescriptlang.org/
https://www.typescriptlang.org/docs/handbook/dom-manipulation.html
https://www.javatpoint.com/typescript-features