**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***