**TypeScript**

**Características de TypeScript:**

* TypeScript es un lenguaje de código abierto que se basa en JavaScript.
* Desarrollado por Microsoft
* El código TypeScript al compilarse se vuelve en código JavaScript.

**Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente**

* TypeScript valida el código (detección de errores), éstos evita que se generen errores en tiempo de ejecución.
* TypeScript como su nombre lo indica, es un lenguaje tipado, es decir, todo tiene un tipo de dato, se basa en el concepto de Orientación a Objetos, Funciones Arrow, etc…
* **S**

**Declaración de variables:**

* **Los nombres de las variables** (llamados identificadores) deben cumplir con las siguientes reglas:
  + El nombre de un identificador debe iniciar con una letra, \_, $,
  + Dentro del identificador se pueden definir dígitos (0-9), letras (minúsculas, mayúsculas)
  + Permite utilizar la mayoría de las letras definidas en los estándares ISO 8859-1 o Unicode( Ejemplo: û, Ü)
  + Además de secuencias de escape Unicode de la forma \uXXXX
* **Case Sensitive**

Importante recordar que TypeScript es CaseSensitive, por ejemplo:

let animal;

let Animal;

Se entienden como dos variables diferentes.

* **Definición de variables (var, let, const)**
  + **var**:
    - Son de un ámbito global.
    - Se les puede aplicar **Hoisting (Elevación de variables)**

Para las variables de tipo **var** primero se usa la variable y después se declare, por ejemplo:

  console.console.log(x);

  var x = 3;

TypeScript hace un barrido de las variables y las pone a inicio al momento de hacer la compilación.

* + **let:** 
    - Son de un ámbito de bloque (únicamente son reconocidas en el bloque en el que fueron declaradas)
  + **const:** 
    - Son de un ámbito de bloque
    - Son constantes, por lo que, SE DEBEN INICIALIZAR con un valor al momento de declararlas, no se puede modificar su valor, pero su estado interno si es modificable (Por buena práctica, se deben escribir con mayúsculas).

**Ejemplo 1:**

const valor = 10;

valor = 15;

En este caso, se intenta cambiar el valor de la constante “valor”, en este ejemplo se genera un error.

**Ejemplo 2:**

    const coche = {

      marca: "VolksWagen",

      submarca: "Tiguan",

      modelo: 2024

    }

    coche = {

      marca: "Ford",

      submarca: "Mustang",

      modelo: 2023

    }

En este primer escenario, se intenta cambiar el valor de coche, y esto genera un error, sin embargo, si es permitido realizar el valor se los atributos de coche.

    const coche = {

      marca: "VolksWagen",

      submarca: "Tiguan",

      modelo: 2024

    }

    coche.marca = "Honda";

    coche.submarca = "CR-V";

    coche.modelo = 2020;

Es decir, el valor interno de la constante coche si se puede cambiar.

* **Tipos de variables**
  + **boolean** 🡪 let bandera: boolean = true;
  + **number 🡪** let entero: number = 6; let decimal: number = 6.5; let hex: number = 0xf00d; let binary: number = 0b1010; let octal: number = 0o744; let big: bigint = 100n;
  + **string 🡪** let cadena: string = “Hola”; let cadena: string = ‘Hola’;
  + **Array 🡪** let numeros: number[] = [1,2,3]; let números: Array<number> = [1,2,3];
  + **Tuplas 🡪** let tupla: [string, number]; tupla[“hola”, 10];
  + **Unknown 🡪 let desconocido: unknown = 4; let desconocido: unknown = “Hola”;**

**Es un comodín que acepta cualquier tipo de valor** (Se utiliza cuando no se tiene conocimiento del tipo de dato que se va a obtener)

* + **any 🡪** let dato: any = “cadena”; dato = 18; dato = true;

**Es un comodín que acepta cualquier tipo de valor.**

* + **void 🡪 Para indicar que una función no retorna ni un valor.**
  + **Si se utiliza en variables, los únicos valores permitidos son:**
    - * let a: void = null;
      * let b: void = undefined;
  + **object 🡪 Para representar todo aquello que no es un tipo de dato primitivo.**

let pais = {“nombre”: “México”, “poblacion”: 123456}

let pais2 = {nombre: string, poblacion: number}

pais2 = {nombre: “México”, población: 654321}

* + **enum 🡪** enum Direccion {Arriba, Abajo, Izquierda, Derecha}

let d: Direccion = Direccion.Izquierda; //Obtiene la posición del elemento

console.log(Direccion[0]); //Obtiene el nombre (cadena) del elemento 0

**enum 🡪** enum Direccion {Arriba = “ARRIBA”, Abajo = “ABAJO”, Izquierda = “IZQUIERDA”, Derecha = “DERECHA”}

console.log(Direccion.Arriba); //Obtiene el valor (ARRIBA)

* **Inicialización de variables**
  + Si no se inicializan con un valor especifico a las variables, estás se inicializan con **undefined**
* **Tipos de funciones**
  + **Funciones nombradas**

function suma(dato1: number, dato2: number): number {

  return dato1+dato2;

}

suma(12, 18);

* + **Funciones anónimas**

let resultadoSuma = function (dato1: number, dato2: number): number {

  return dato1+dato2;

}

resultadoSuma(2,16);

* + **Funciones Arrow**

Simplifican la forma de implementar funciones.

let sum = (dato1: number, dato2: number) => dato1+dato2;

* **Tipos de parámetros**
  + **Parámetros opcionales**

Los parámetros opcionales siempre se ponen al final de la lista de parámetros.

Este tipo de parámetros se denotan de las siguientes dos formas:

* + - **Van precedidos por un “?” (signo de interrogación).**

function saludar(msg1: string, msg2?: string): void {

  console.log(msg1+" "+msg2);

}

* + - **Se deben inicializar con un valor.**

function saludar(msg1: string, msg2: string = ""): void {

  console.log(msg1+" "+msg2);

}

* + **Parámetros de tipo Rest**

Para indicar que se tiene la necesidad de trabajar con un número indeterminado de parámetros.

function saludar(...msgs: string[]): void {

  console.log(msgs);

}