# TypeScript
# TIPOS DE DATOS

# TIPOS DE DATOS

- Son opcionales, **si no se ponen, los infiere:**

```
let i = 0;  //infiere number
funcion f() {return 'hola';}  //infiere string
```

- Ej. definición de una variable:

```
var nombre: string;
```

- Ej. declaración una función:

```
function saludar (nombre: string): string {
        return "Hola " + nombre;
}
```

# TIPOS DE DATOS

- Tipo string:

**var direccion: string;**

  - Son textos
  - Una línea van entre “ “ ó ‘ ‘;
  - Varias líneas:
    - “ linea 1 \n linea 2”
    - `linea 1
      linea 2`
  - **Plantillas texto**: permiten incrustar valores de propiedades o variables dentro de una cadena de texto:

**${expresión}**

Ejemplo:

`Mi nombre es ${nom} y mi edad ${edad}`

# TIPOS DE DATOS

- Ejemplos de plantillas string (probadlos en PlayGround):

```
var template = `
    <div>
    <h1>Hello</h1>
    <p>This is a great website</p>
    </div>
    `
```

# TIPOS DE DATOS

- Más potencia de las plantillas string (Templates string):
  - **Admiten incluso cualquier expresión o función JavaScript dentro de { }.**
  - Ejemplo:

```
 let cadena:string=`El resultado de 3+2 es: ${3+2} `
console.log(cadena);
```

```
function getEdad():number{
return 40;
}

let cad:string;
cad=`Tu edad es: ${getEdad()}`
console.log(cad);
```

# TIPOS DE DATOS

- Ejemplos string para probar en PlayGround:

```
var nombre: string;
nombre = "Pedro";    //usando " "
nombre = 'Lucas';    //usando ' '
var edad: number = 3;

var cadena: string = `Pedro tiene ${edad} años`;  //usando plantilla de
                                                  texto ${ }
alert(cadena);

var cadena2: string = "Pedro tiene \n 33 años";  //cadena con 2 líneas
alert(cadena2);

var cad: string = `estoy en la linea 1
                   ahora la 2
                   y la 3`;        //cadena con 3 líneas
alert(cad);
```

# TIPOS DE DATOS

- Tipo number:

**var edad: number;**

- Cualquier tipo de número
- Todos los números se representan internamente en punto flotante

- **CONSTANTES ESPECIALES**:

a) **NaN** (Not a Number): algo no es un número.

**Ej:**

```
var dayOfMonth = 50;
if (dayOfMonth < 1 || dayOfMonth > 31) {
        dayOfMonth = Number.NaN;
        alert("Day of Month must be between 1 and 31.");
}
```

Otro:

```
> Number("xyz")
NaN
```

# TIPOS DE DATOS

- **CONSTANTES ESPECIALES** continuación:

b) **Infinity** (infinito): por ejemplo, división por 0 no da error, da Infinity.

**Ejemplos:**

```
> 3/0
Infinity

> Infinity + Infinity
Infinity

> Infinity - Infinity
NaN

> 5 * Infinity
Infinity
```

# TIPOS DE DATOS

- Tipo boolean:

**var casado: boolean;**

  - Valores true o false

**Ej:**

```
let a: boolean = true;
let b: boolean = false;
let c: boolean = 23; // Error
let d: boolean = "blue"; // Error
```

# TIPOS DE DATOS

- Tipo Date:

**let hoy: Date;**

  - Valores de fecha

    **Ej:**

```
let hoy: Date= new Date();
hoy=new Date('2018-12-23');
```

# TIPOS DE DATOS

- Tipo Array:
  - Dos formas de declararlos

    **Ej:**

```
var jobs: Array<string> = ['IBM', 'Microsoft', 'Google'];
var jobs: string[] = ['Apple', 'Dell', 'HP'];

var chickens: Array<number> = [1, 2, 3];
var chickens: number[] = [4, 5, 6];
```

# TIPOS DE DATOS

- Tipo Tupla:
  - Permite expresar un ARRAY donde **el tipo de un nº fijo de elementos es conocido**, y no tiene por qué ser el mismo para todos.
  - Una vez definidos los tipos conocidos, **el resto de elementos del array** deben ser obligatoriamente de **esos tipos.**

**Ej:**

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
// Initialize it incorrectly
x = [10, "hello"]; // Error
```

# TIPOS DE DATOS

**MÁS EJEMPLOS:**

```
let a: [number, string] = [11, "monday"];
let b: [number, string] = ["monday", 11]; //
Error
let c: [number, string] = ["a", "monkey"]; //
Error
let d: [number, string] = [105, "owl", 129, 45,
"cat"];  //da warning por length pero funciona
let e: [number, string] = [13, "bat",
"spiderman", 2]; //da warning por length pero
funciona

e[13] = "elephant";   //Lo inserta en la
posición 13
e[15] = false; // Error
```

# TIPOS DE DATOS

## CURIOSIDADES DE LOS ARRAYS EN JS y TS:

- Siempre que accedemos a un elemento cuyo índice está fuera del rango del array, **NO DA ERROR:**
  - crea esa posición e inserta ese elemento, o
  - si es una consulta, devuelve "undefined"

**Ejemplo para probar en PLAYGROUND:**

```
var array: number[ ] = [3, 4, 5, 6];

document.body.innerHTML += "<p>Posición 10: " + array[10] + "</p>";
array[10] = 88;
document.body.innerHTML += "<p>Elemento en pos 10 añadido: " +
array[10] + "</p>";
document.body.innerHTML += "<p>Posición 0: " + array[0]+"</p>";

for (var i = 0; i < array.length; i++){
    document.body.innerHTML += "<p>Pos i:" + i + "=" + array[i]+"</p>";
}
```

# TIPOS DE DATOS

- Tipo object:
  - Permite definir una variable como un tipo de objeto concreto.
  - Para ello, indicaremos entre llaves qué atributos o propiedades tendrá ese objeto y de qué tipo serán.

EJ :

```
let obj: {a: string, b:number}; //indicamos que los objetos que se asigne a la variable obj tienen que tener un atributo llamado "a" que es de tipo string y otro llamado "b" de tipo number.

obj= {a: "hola", b: 33}; //ok

obj={a:33, b:"adios"}; //error, tipos erróneos
```

# TIPOS DE DATOS

- Tipo enum o Enumerados:
  - Permite dar nombre a un conjunto de valores NUMÉRICOS.

    EJ 1: Si queremos crear una lista de roles de persona:

```
enum Role {Employee, Manager, Admin};
var role: Role = Role.Employee;
```

  - Por defecto, **valor inicial de un enumerado**=0, se puede cambiar así:

```
enum Role {Employee = 3, Manager, Admin};
var role: Role = Role.Employee;
// en este ejemplo, Manager=4 y Admin=5
```

  - Podemos dar valores personalizados a cada enum:

```
enum Role {Employee = 3, Manager=5, Admin=7};
var role: Role = Role.Employee;
```

# TIPOS DE DATOS

- También podemos darle el VALOR de un elemento y nos dirá su NOMBRE.

  EJ 1: Si queremos saber el color con valor 2 en este enumerado:

```
enum Color {Red = 1, Green, Blue}
let colorName: string = Color[2];
alert(colorName);
// Displays 'Green' as it's value is 2 above
```

# TIPOS DE DATOS

- Tipo any:
  - Es el tipo por defecto si no ponemos nada
  - Una variable de tipo any admite cualquier valor y llamar a cualquier método, exista o no.
  - **Tipo Object es más restrictivo**: sólo llamar métodos que define él.

**EJEMPLOS:**

```
let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false; // okay, definitely a boolean

let notSure: any = 4;
notSure.ifItExists(); // okay, ifItExists might exist at runtime
notSure.toFixed(); // okay, toFixed exists (but the compiler doesn't check)
let prettySure: Object = 4;
prettySure.toFixed(); // Error: Property 'toFixed' doesn't exist on type 'Object'.
```

# TIPOS DE DATOS

- Cuando usar any???

  1. Cuando esperamos valores que vengan del usuario o de aplicaciones de terceros (Internet, por ej.) y **no sabemos de qué tipo pueden ser**.
  2. Cuando **tenemos cierta idea de qué tipo va a ser,** pero no lo tenemos del todo identificado. Por ejemplo, sabemos que es un array, pero no tenemos claro de qué tipo. Haríamos:

```
var list: any[] = [1, true, "free"];
list[1] = 100;
```

# TIPOS DE DATOS

- Tipo void:
  - Usar void significa: "**no hay tipo esperado**".
  - Normalmente **se usa en funciones** (métodos) que no devuelven valor.

**EJEMPLO:**

```
function warnUser(): void
{
    alert("This is my warning message");
}
```

  - Declarar variables de tipo Void **no tiene sentido**, ya que sólo podríamos asignarles el valor **undefined** o **null** a ellas.

**EJEMPLO:**

```
let unusable: void = undefined;
```

# TIPOS DE DATOS

- Tipo null:
  - Existe el **tipo null** y el **valor null**:

```
var x:null = null;  //x:null  es tipo;   = null   es  valor

var y:number=null;   //null es valor
```

**NOTA**: cuando el indicador **strictNullChecks** se configura como true en **tsconfig.json**, solo el valor null se puede asignar a las variables con tipo null.
Así :

```
var y:number = null; //dará ERROR
```

Este indicador está **desactivado por defecto**, lo que significa que también ***puede asignar el valor null a variables con otros tipos como number o void.***

*(VER ENLACE)*

# TIPOS DE DATOS

- Tipo undefined (indefinido):
  - Cualquier variable cuyo valor no haya especificado se establece en **undefined.**
  - Existe el **tipo undefined** y el **valor undefined**:

```
var x:undefined = undefined;

        //x:undefined es tipo;   = undefined es valor

var y:number=undefined;   //undefined es valor
```

**NOTA**: cuando el indicador **strictNullChecks** se configura como true en **tsconfig.json**, solo el valor undefined se puede asignar a las variables con tipo undefined.
Así :

```
var y:number = undefined; //dará ERROR
```

Este indicador está **desactivado por defecto**, lo que significa que también *puede asignar el valor undefined a variables con otros tipos como number o void.*

*(VER ENLACE)*

# TIPOS DE DATOS

- Tipo never:
  - Representa el tipo de valores que nunca ocurre.
  - **Por ejemplo: never** podría ser el **tipo devuelto por una función que nunca vuelve** (*bucle infinito)* o que siempre lanza una excepción.

**EJEMPLOS:**

```
// Function returning never must have unreachable end point
function error(message: string): never
{
 throw new Error(message);
 }
// Inferred return type is never
function fail() {
    return error("Something failed");
 }
// Function returning never must have unreachable end point
function infiniteLoop(): never {
                while (true) { }
 }
```

# TIPOS DE DATOS

- Tipo Aserciones (≈ cast en Java):
  - Se parece al cast de otros lenguajes, con la diferencia de que no chequea tipos de datos: SE FÍA DE NOSOTROS.
  - **Por ejemplo, si tecleamos:**

```
var nombre:any=123; //tipo numérico
var nuevoNombre:string = <string> nombre;
//no da error, se fía de nosotros.
alert(nuevoNombre.length); //muestra undefined, pero no da error.
```

  - **HAY 2 FORMAS DE INDICARLO:**

a) **Usando <tipo de cast>:**

```
let someValue: any = "this is a string";
let strLength: number = (<string>someValue).length;
```

b) **Usando as:**

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```