



Sintaxis de un lenguaje de programación

Lenguaje TypeScript

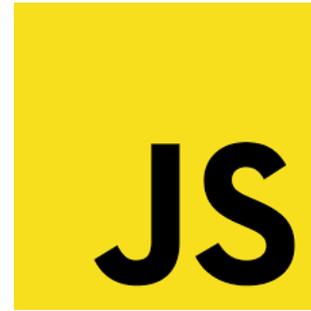
<https://www.typescriptlang.org/>



Juan Vladimir
@juanvladimir13

Que es Deno

Is runtime for **TypeScript** and **JavaScript**. Features built-in dev tools, powerful platform APIs, and native support for TypeScript and JSX.



<https://www.typescriptlang.org/>

<https://developer.mozilla.org/es/docs/Web/JavaScript>

Crear archivo de texto plano TypeScript

Ingresar al **CMD** o **Terminator**

Crear un **archivo de texto plano** con la extensión ***.ts**

Ejecutar un archivo TypeScript

Verificar errores de sintaxis, ejecutar `deno lint nombreDeArchivo.ts`

Formatear el código, ejecutar el comando `deno fmt nombreDeArchivo.ts`

Ejecutar el código, ejecutar el comando `deno run nombreArchivo.ts`

Tipos de datos

Números enteros (positivos, negativos)

temperatura = -13

anio = 2025

Números con decimales (positivos, negativos)

peso = 60.48

latitud = -18.755121545

Texto / Cadenas / Caracteres / String

nombres = 'Juan Vladimir'

carnet = '12345678'

Booleanos

perteneceAlCurso = true

esMayorDeEdad = false

Variables, constantes y asignación

Variable

```
let edad = 36;           // variable de tipo de dato numero
let nombre = 'juanvladimir13'; // variable de tipo de dato string
let isDeveloper = false; // variable de tipo de dato boolean
```

Constante

```
const FECHA_NACIMIENTO = '26-12-1987';
const CANTIDAD_BOMBAS_MAX = 20;
```

Asignación

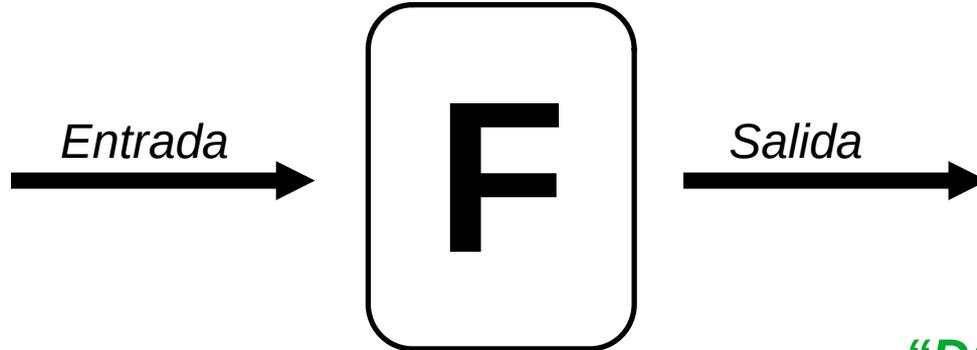
```
edad = 36;
nombre = 'Ing. Juan Vladimir';
isDeveloper = true;
```

Función

Es un **bloque de código** reutilizable que realiza una tarea específica.

Las funciones permiten **estructurar el código en partes más pequeñas y manejables**, lo que facilita su **mantenimiento y reutilización**.

Un **algoritmo** se resuelve con *una o muchas funciones*



“Divide y vencerás”

Partes de una Función

Nombre: Se usa para "llamarla" o invocarla. Este nombre debe ser descriptivo para indicar qué hace la función.

Parámetros: Son **ceros, uno o más valores de entrada**, que la función utiliza para realizar su tarea.

Cuerpo: Es el conjunto de instrucciones que se ejecutan cuando la función es llamada.

Valor de retorno: Valor como resultado de la tarea. Este valor puede ser utilizado por el código que llamó a la función.

Llamada a la función: Para usar una función, se "llama" por su nombre seguido de paréntesis. Si la función requiere parámetros, estos se colocan dentro de los paréntesis.

```
function calcularEdadDeEstudiante( nombre:string, anioNac:number ):number {  
    // Cuerpo (sentencias)  
    return 0;  
}
```

```
$edad = calcularEdadDeEstudiante( 'juanvladimir', 2005 );
```

Tipos de dato

Números

```
let edad : number = 13;  
function sumar(numX: number, numY: number): number { return 0; }
```

Cadenas

```
const fechaNacimiento : string = '26-12-1987';  
function saludo(usuario: string): string { return 'Hola'; }
```

Booleanos

```
const hasGirlFriend : boolean = false;  
function aceptasSerMyLove(persona: string): boolean { return false; }
```

Tipo de dato de retorno void

```
function destruirElMundo(cantidadBombas: number): void { }
```

Operadores Aritméticos

let sueldo : **number** = 3200;

Suma	+	sueldo = sueldo + 5;
Resta	-	sueldo = sueldo - 5;
Multiplicación	*	sueldo = sueldo * 5;
División	/	sueldo = sueldo / 3;
Residuo	%	sueldo = sueldo % 3;
Exponente	**	sueldo = sueldo ** 3;
Incremento	++	sueldo++ ;
Decremento	--	sueldo--;

Operadores Lógicos

Conjunción: AND &&

Disyunción: OR ||

Negación: NOT !

```
let seComportanBien : boolean = true;
```

```
let hacenTareas : boolean = true;
```

```
let hayChurrasco : boolean = false;
```

```
hayChurrasco = seComportanBien && hacenTareas;
```

```
hayChurrasco = seComportanBien || hacenTareas;
```

```
hayChurrasco = !hayChurrasco;
```

Operadores de comparación

```
let sueldoGerente : number = 3200;  
let sueldoEmpleado : number = 5500;
```

Igual	==	sueldoGerente == sueldoEmpleado;
Diferente	!=	sueldoGerente != sueldoEmpleado;
Igual estricto	===	sueldoGerente === sueldoEmpleado;
Diferente estricto	!==	sueldoGerente !== sueldoEmpleado;
Mayor que	>	sueldoGerente > sueldoEmpleado;
Menor que	<	sueldoGerente < sueldoEmpleado;
Mayor igual	>=	sueldoGerente >= sueldoEmpleado;
Menor igual	<=	sueldoGerente <= sueldoEmpleado;

El resultado de una **comparación** siempre es una **proposición** *false* o *true*

Expresión

VAR : variable

FUN : función

OPE : operador/conector

CON : constante

Expresión unaria

OPE { **VAR** | **FUN** | **CON** }

Ejemplos:

a = ~hayChurrasco;

res = calcularSuma(8, 5);

curso = 'Quinto D';

Expresiones

{ **VAR** | **CON** | **FUN** } **OPE** { **VAR** | **CON** | **FUN** }

Ejemplos:

x = 8 + 5;

y = sueldoGerente > sueldoEmpleado;

z = calcularSuma(5,5) * 0.3;

a = haceTareas && esLinda;

b = haceTareas && (sueldo > 3000);

c = (sueldo + 1500) - gastos;

Sentencia

Es una **línea de código** que llevan a cabo una tarea. Las sentencias están formadas por un **conjunto de expresiones** que permiten realizar una acción determinada.

```
let seComportanBien : boolean = true;
```

```
let hacenTareas : boolean = false;
```

```
let hayChurrasco : boolean = false;
```

```
let sueldo : number = 3200;
```

```
let sueldoGerente : number = 5000;
```

```
hayChurrasco = seComportanBien && hacenTareas;
```

```
y = sueldoGerente > sueldoEmpleado;
```

```
z = calcularSuma(5,5) * 0.3;
```

Estructuras de control

Estructura de control condicional (if else)

Ejecuta una **sentencia si una condición específica es evaluada como verdadera**. Si la condición es evaluada como **falsa**, otra sentencia puede ser ejecutada.

```
let sueldo : number = 3200;  
let hacenTareas : boolean = true;
```

Sintaxis

```
if ( sueldo > 3200 ) {  
    // sentencias o bloque de código  
}
```

```
if ( hacenTareas == false ) {  
    // sentencias o bloque de código  
}
```

```
if ( sueldo > 3200 && hacenTareas ) {  
    // bloque de código por sentencia true  
} else {  
    // bloque de código por sentencia false  
}
```

```
If ( (sueldo - 1000) > 3200 ) {  
    // sentencias o bloque de código  
}
```

Estructuras de control

Estructura de control repetitiva (while)

Ejecuta una **sentencia especificada mientras cierta condición se evalúe como verdadera**. Dicha condición es evaluada antes de ejecutar la sentencia.

```
let n : number = 0;
let estaLloviendo : boolean = true;
```

Sintaxis

```
let n: number = 0;
```

```
while (n > 13 ) {
  // sentencias
  $n++;
}
```

```
while ( estaLloviendo ) {
  // sentencias
}
```

Estructuras de control

Estructura de control repetitiva (for)

Expresión inicial: Declaración de variable y asignación

Condición: Expresión para ser evaluada antes de cada iteración del bucle. Si esta expresión se evalúa como verdadera, se ejecuta sentencia, si es falsa, la ejecución salta a la siguiente expresión.

Expresión final: Expresión evaluada al final de cada iteración del bucle. Esto ocurre antes de la siguiente evaluación de la condición. Generalmente se usa para actualizar o incrementar la variable contador.

Sintaxis

```
for (let index=0; i < 13; i++) {  
    // sentencias  
}
```

```
foreach (dataList as value) {  
    echo $value;  
}
```

```
foreach ($dataList as $value) {  
    echo $value;  
}
```

Operadores de Asignación Avanzados

```
let sueldo : number = 3200;
```

```
let hayChurrasco : boolean = false;
```

Suma	+=	sueldo += 5;
-------------	-----------	--------------

Resta	-=	sueldo -= 5;
--------------	-----------	--------------

Multiplicación	*=	sueldo *= 5;
-----------------------	-----------	--------------

División	/=	sueldo /= 3;
-----------------	-----------	--------------

Residuo	%=	sueldo %= 3;
----------------	-----------	--------------

Exponente	**=	sueldo **= 3;
------------------	------------	---------------

AND	&&=	hayChurrasco &&= sueldo > 3000;
------------	--------------------	---------------------------------

OR	 =	hayChurrasco = sueldo > 3000;
-----------	------------	---------------------------------