MIGUEL RODRÍGUEZ POLO |2º DAW | 1º HITO INDIVIDUAL

CUESTIÓN 1 A) La principal característica de Typescript es el tipado estático. Decimos que un lenguaje es de tipado estático cuando cumple con estas características principales:

* Las variables tienen un tipo de dato.
* Los valores sólo se pueden asignar a variables del tipo correspondiente.

let edad : number; //Asignamos el tipo number para la variable edad

edad = 20; // La variable ahora sólo puede asignar valores del tipo number

A partir de estas dos características principales, se derivan algunas otras, como por ejemplo:

* Interfaces
* Genéricos
* Casting de datos (conversión de tipos)
* Argumentos con tipo
* Tipo de retorno para las funciones
* Mucho más

El contraste de estos lenguajes son los de tipado dinámico, como JavaScript, estos lenguajessuelen ser mucho más flexibles, lo que nos permite escribir código menos verboso.

# B) Tipos de variables

TypeScript pertenece al grupo de lenguajes imperativos pero, sin embargo,

se caracteriza por estructuras de tipado débil donde la declaración de

variables no exige la asociación con un tipo de datos de forma implícita y unívoca.

TypeScript tiene un tipo de datos dinámico. Esto significa que una variable

puede declararse conteniendo un tipo de dato y, más adelante, asignarle otro tipo mediante una simple asignación. Ejemplo:

var cambioTipo = "hola"; //es un string

cambioTipo = 89; //Ahora es un número

# Numéros (Number)

Números enteros y decimales, positivos y negativos.

En otros lenguajes de programación se distingue entre distintos tipos de números (int: entero, float: decimal…), por lo que hay que indicarlo al declarar la variable, pero en TypeScript no.

var numero = 5; var numero\_decimal = 4.1;

var numero\_negativo = -12;

¿Que ocurre al intentar sumar decimales con enteros? ( 2.1 + (-12))

|  |  |
| --- | --- |
|  | numero\_decimal + numero\_negativo |
| Resultado = -7.9  El resultado es el esperado. Los números en TypeScript funcionan y se convierten de forma automática. |

# Cadenas (String)

Una o varias cadenas de caracteres alfanuméricos (letras, números, signos puntuación…).

Las denominamos tipos de datos o variables de texto porque las empleamos para almacenar información de esa naturaleza o porque sus valores los vamos a tratar como texto. Ejemplo:

var nif = "12332112W",  ciudad= "Santiago de Compostela",

codigoPostal = "28001",

telefono = "003423145627";

ocupacion = "Estudiante de Biología";

Como vemos en los ejemplos, el código postal o el teléfono, aunque contengan solo números, no nos interesa tratarlos de esa forma (en el

sentido de aplicarles operaciones matemáticas: sumarlos, restarlos,

comparar si son mayores o menores…). Tiene más sentido tratarlos como cadenas de texto, por lo que asignamos su valor entrecomillado.

## Pruebas cruzadas con concatenaciones de String y Number

var numTexto1= "3";

//variable tipo string-texto al asignar el valor entrecomillado var numTexto2= "5"; //igual tipo que la anterior document.write("Al usar el operador + con las variables string 3 y 5 obtenemos: "); document.write(numTexto1 + numTexto2);

//te aparecerá la cadena "35”, al haber sido tratadas como texto var num1 = 3 ;

//variable tipo number-número al asignar el valor sin comillas var num2 = 5; //igual tipo que la anterior document.write("<br>");

document.write("Al usar el operador + con las variables number 3

y 5 obtenemos: ");

document.write(num1 + num2);

//te aparecerá el resultado de la suma: 8, al haber sido tratado como números

# Booleanos (Banderas,condiciones...etc)

El nombre viene de la lógica de Boole. Pueden tomar solo dos valores:

‘true‘ o ‘false‘ (verdadero/falso).

Se utilizan mucho en las sentencias condicionales, a modo de interruptor, pudiendo equiparar los valores true/false, en ese símil, a ‘encendido’ /apagado’, es decir, que hace que el programa se siga ejecutando por una instrucción o por otra diferente:

*“Si es verdad, sigue ejecutando X, si es falso, haz Y”.*

var boolean = false if(boolean ==true){

//esto no ocurre

}

if(boolean ==false){

//esto si ocurre

}

Estas variables no deberían combinarse con el resto, ya que están mayormente diseñadas para ser usadas en condiciones.

var bool1 = true; //variable tipo number-número al asignar el valor sin comillas  var num2 = 5; //igual tipo que la anterior

document.write("<br>");

document.write("Al usar el operador + con las variables

boolean en true y number 5 obtenemos: ");  document.write(bool1 + num2); //No deja compilar

A diferencia de JavaScript, TypeScript no permite este tipo de conversiones.

# List (Listas)

Los Arrays ó matrices pueden contener múltiples valores, como un buzón

con diversos compartimentos numerados en su interior. Son del tipo objetos:

|  |
| --- |
| var granPoblacion = ["Madrid", "Barcelona", "Valencia", |
| "Sevilla"]; |

Hay dos principales elementos en su estructura:

* **corchetes [ ]:** para incluir los valores
* **comas (,):** para separar los valores dentro del array

En el array, los valores (que pueden ser numéricos, de texto o una combinación de ellos) se encuentran en posiciones numeradas, empezando a contar desde el cero, no desde el uno. Es decir, los valores de cada

elemento del array (a los que se accede con nombreArray[n], siendo n=

0,1,2…) serían:

granPoblacion[0] = Madrid granPoblacion[1] = Barcelona granPoblacion[2] = Valencia

granPoblacion[3]= Sevilla

Podrías añadir o cambiar valores dentro del array. Por ejemplo, la

sentencia: granPoblacion[4] = "Zaragoza"; añadiría este valor detrás del

valor ‘Sevilla’. Sin embargo es preferible el método push()

Ahora voy a construir esta lista leyendo el array uno a uno

granPoblacion.forEach(function(ciudad){ document.write("<li> " + ciudad + "</li>");

})

Las listas no se pueden sumar o concatenar con otras variables de forma primitiva. Ahora voy a intentar sumarle 3 a la lista

Resultado: Madrid,Barcelona,Valencia,Sevilla3

Lo que ocurre es que el 3 simplemente se concatena. La lista se ha impreso

ordenada con todos sus elementos. C)

# Diferencias var, let y const

TypeScript te obliga a elegir un tipo de variable. Ya sea var, let o const.

nombre = "hola"

# Var contra Let

El problema de las variables declaradas con var es que se comportan diferente que en la mayoría de lenguajes de programación, que limitan el alcance(o contexto) por bloque, es decir, entre llaves ({ y }). Esto no ocurre en JavaScript cuando utilizamos var. Por ejemplo:

|  |
| --- |
| if (true) { var color = "negro"  } console.log(color) // "negro" |

Con var, la única forma de limitar el alcance es dentro de una función:

function miFuncion() { var color = "negro"

}

console.log(color) // error

Esto puede llegar a ocasionar una gran cantidad de problemas, entre ellos la sobreescritura

|  |
| --- |
| if (orange === 'orange') { var orange = 'blue'; // el ambito es global let apple = 'green'; // el ambito está dentro del bloque IF  console.log(orange); // azul console.log(apple); // verde  } console.log(orange); // azul console.log(apple); // rojo |

En este ejemplo, el valor original de la variable orange se pierde completamente en la función debido a la sobreescritura. TypeScript no sabe distinguir de contexto local o global cuando se trabaja con var.

A diferencia de var, let trabaja siempre en su bloque, que puede ser un if, un for, un while...etc. Esto impide que se declaren variables let completamente globales, y por lo tanto impide los riesgos derivados.

Hoy en dia, se recomienda encarecidamente utilizar let lo máximo posible por estas razones.

# Const

Const es una variable let, pero con 2 peculiaridades

Debe inicializarse. No puede declararse sin valor

const direccion = "calle paco"

Una vez se le asigna un valor inicial, no puede ser modificado

|  |  |
| --- | --- |
|  | const color\_manzana = "rojo" color\_manzana="verde" //error |
| Const es un tipo de variable intransigente, cuyo contenido no puede ser modificado. Su utilidad reside en la memoria que se ahorra al utilizarla.  Cuando declaras una variable como let o var, estas reservando una cantidad  X de memoria, para los posibles valores que pueda albergar esa variable (Que pueden llegar a ser muy elevados si el programador lo desea). Sin embargo, al usar const, el programa asigna automáticamente solo la memoria necesaria para ese valor. |
| CUESTIÓN 2 |  |
| CUESTIÓN 3 |  |