TYPESCRIPT

Variables var y let

La diferencia está en el ámbito o scope que se estable con cada tipo.

var a = 5;

var b = 10;

if (a === 5) {

  let a = 4; // El alcance es dentro del bloque if

  var b = 1; // El alcance es dentro de la función

  console.log(a);  // 4

  console.log(b);  // 1

}

console.log(a); // 5

console.log(b); // 1

con let se restringe el ámbito.

Constantes

Las constantes tendrán el mismo ámbito que una variable let. Solo está permitada la asignación en su declaración. Pero sí podremos alterar un miembro de la constante:

const options = { frecuency: 10, all: null };

options.frecuency = 20; // Ok

options = { frecuency: 20, all: true } // Error

Operadores

Operadores == y ===

Con === se compara también el tipo de las variables, con == se hace una transformación o cast interno.

let a = new String("cadena");

let b = new String("cadena");

a === b; // false

a == b; // false

La evaluación es falseporque está comparando objetos, que son objetos referencia. Y esta comparación se hace en base a la dirección de memoria. Si las direcciones son iguales, los objetos también lo son. En el ejemplo hemos instanciado dos veces la clase String por lo que tenemos dos objetos con direcciones distintas. Para que la igualdad se pueda evaluar como true: let a = new String("cadena");

let b:String = a;

a === b; // true

a == b; // true

Operador in

Busca el valor del operando de la izquierda, que debe ser any, string o number, en el operando de la derecha, que debe ser any, object o array devolviendo true si lo encuentra y false en caso contrario. No busca valor de los atributos, sino el nombre de los mismos.

let animales = ["Perro", "Gato", "Conejo"];

"Perro" in animales; /\* False. Perro es un valor del íncide 0 del array \*/

1 in animales; /\* true. El array contiene 3 valores con lo cual existe el índice 1 \*/

let obj = {animal:"Perro"}

"Perro" in obj; /\* False. Perro es un valor del íncide “animal” \*/"animal" in obj; // True. Animal es un índice

Control de flujo

switch

let x = 1;

switch (x) {

case 0:

case 1:

case 2:

// Hacer algo si x vale 0, 1 ó 2

break;

default:

/\* Hacer algo en el caso de que no sea ninguno de los anteriores \*/

break;

}

While

let x = 0;

while (x < 10) {

x++; // x va incrementándose en 1 en cada iteración

}

for

for (declaración; condición; actualización) {

// Código

}

for-in

Va volcando el índice de nombres de la variable temporal indice. La ventaja es que no importa de qué tipo sea el índice: numérico, cadena de caracteres, etc. Existe un problema al recorrer arrays de esta forma. Debido a que recorre todas las propiedades de array, es posible que incluso itere por las intrínsceas (como length) lo que daría lugar a comportanmientos inesperados. Para ello debe debe usarte una precondición dentro del bucle que nos asegure de que exista:

let nombres: Array<string> = ["Carlos", "José", "Lama"];

for (let indice in nombres) {

if ( nombres.hasOwnProperty(indice) ){

console.log(nombres[indice]);

}

}

for-of

En la interaction obtenemos el valor no el índice como con for-in.

let names = ["Carlos", "José", "Lama"];

for (let name of names) {

console.log(name);

}

Si compilamos para ES6, es necesario que la estructura que estemos iterando implemente un método Symbol.iterator

Tipos primitivos y objetos

Los tipos primitivos son los elementales que nos proporciona un lenguaje de programación y cuyos valores son guardados en la posición de memoria que se le asigna. A su vez tenemos los tipos objeto que no son más que clases que se han construido alrededor de estos primitivos para dotarlos de más funcionalidades.

string – String

boolean – Boolean

number – Number

undefined – Undefined

null – Null

void –Void

symbol –Symbol

Si declaramos una variable con un tipo primitivo, no la podemos inicializar con el tipo objeto aunque sí se puede hacer lo inverso.

let variable1: Boolean = new Boolean(true); // Correcto

let variable2: boolean = variable1; // Incorrecto boolean/Boolean

let variable3: boolean = true; // Correcto

let variable4: Boolean = variable3; // Correcto