Tipos de datos en JavaScript

En JavaScript cuando se crea una variable no es necesario indicar el tipo de dato que contiene, debido a que el intérprete del navegador se encarga de deducir el tipo de dato en función del valor que hayamos asignado. Esto significa que JavaScript es un "*lenguaje tipado dinámicamente*", y gracias a esto a cualquier variable se le puede asignar (y reasignar) valores de todos los tipos. Por ejemplo, una variable puede contener un dato de tipo numérico y posteriormente almacenar un texto, un booleano, etc.:

```
let x = 123; // Tipo de dato numérico

x = "hola"; // Ahora es una cadena de caracteres
x = true; // Ahora es un booleano
```

Estructuras y tipos de datos

En JavaScript los tipos de datos se pueden clasificar en dos categorías:

- **Tipos de datos primitivos:** aquellos que no tienen propiedades ni métodos, y son inmutables, es decir, son valores que no se pueden cambiar

- **Tipos de datos estructurados o de referencia**: se utilizan para almacenar colecciones de datos y entidades más complejas, pueden contener múltiples valores y tienen propiedades y métodos.

El siguiente esquema muestra un resumen sobre los tipos de datos en JavaScript:

		Nombre	Descripción
TIPOS DE DATOS EN JAVASCRIPT	**Tipos de datos primitivos**	String	Cadenas de texto
		Number	Valores numéricos
		Boolean	Valores de verdadero o falso
		BigInt	Valor numérico para representar enteros grandes (1234567890123456789n)
		Symbol	Un valor primitivo único e inmutable.
		Undefined	Representa una variable que no ha sido declarada o a la cual no se le ha asignado un valor.
	Tipos de datos estructurados	Tipos predefinidos de JavaScript	- Object: Es el objeto base en JavaScript y todos los demás objetos son prototipos del mismo. - Function: las funciones también se consideran objetos y tienen métodos y propiedades. - null: aunque técnicamente no es un objeto, `typeof null` devuelve "object
		Tipos especiales	Array, Date, RegExp, Map, Set, Promise, WeakMap, WeakSet, Math, JSON, otros.
		Otros	Objeto global
			Objeto prototipo
			Otros

Tal y como indica el esquema anterior, los tipos de datos primitivos son seis:

- *Symbol.*
- Bigint.
- *Undefined.*
- *Boolean.*
- *Number.*
- *String.*

Todos los datos primitivos son inmutables, y esto significa que no se pueden modificar, es decir, que se puede reasignar un nuevo valor a una variable, pero el valor existente no se puede cambiar, contrario a como sucede con los objetos, los arreglos y las funciones, los cuáles si se pueden modificar.

Los tipos de datos primitivos son controlados por el operador *typeof*, que tal y como vimos en otras lecciones de este módulo, sirve para determinar el tipo de dato que almacena una variable. Si deseamos verificar cualquier tipo estructural derivado de *Object*, en lugar de utilizar *typeof* debemos emplear el operador *instanceof* que sirve para comprobar qué tipo de Objeto estamos usando.

A continuación, se explica brevemente para que se utilizan los diferentes tipos de datos y estructuras en JavaScript.

Tipo Number

Se utilizan para almacenar valores numéricos enteros (llamados *integer* en inglés) o decimales (llamados *float* en inglés). Los números decimales utilizan el carácter punto en vez de coma para separar la parte entera y la parte decimal.

```
let n = 4589;
 n = 4.589;
```

Además, existen los llamados “valores numéricos especiales”:

Infinity: representa el Infinito matemático. Se obtiene como resultado de la división por cero, o simplemente hacer referencia a él directamente::

```
let a = 1;
let b = 0;
let r = 1 / 0;
console.log(r);
```

NaN (*Not a Number*): representa un error de cálculo como resultado de una operación matemática incorrecta o indefinida:

```
let x = 100;
let z = "manzana";
let resultado = x / z;
console.log(x / z)
```

Tipo String

También llamados cadenas de caracteres o cadenas de texto, se utilizan para almacenar caracteres alfanuméricos, tales como palabras o frases, cifras, incluso espacios en blanco.

Los *strings* deben estar encerrados entre comillas simples, comillas dobles o *backticks* (acento grave) o de lo contrario, JavaScript intentará interpretarlos como otro nombre de variable. Esto significa que todo lo que se coloca entre comillas es tratado como una cadena de texto, independientemente de lo que coloquemos en el interior de las comillas.

```
let mensajeBienvenida = "Bienvenido(a) a nuestro sitio web";
let nombreDeProducto = 'tablet';
let valor = "550";
let espacio = " ";
```

En el ejemplo anterior, 550 se considera como una cadena de texto, debido a que está encerrado entre comillas. Lo mismo el valor de la variable espacio, que aparentemente está vacío, pero en realidad también se considera como una cadena de caracteres.

En los casos en los que necesitemos utilizar comillas en el texto de un string, se recomienda emplear comillas diferentes a las que definen la cadena. Por ejemplo:

```
// comillas dobles dentro de comillas simples
let frase = 'Para obtener el descuento necesitas la tarjeta de "Cliente preferencial"';

// comillas simples dentro de comillas dobles
let frase2 = "Debes introducir una contraseña con caracteres alfanuméricos, ejemplo 'beta1235'";
```

Los strings son inmutables

A diferencia de algunos lenguajes de programación (por ejemplo lenguaje C), las cadenas de caracteres o de texto en JavaScript son inmutables. Esto significa que una vez que se crea una cadena, no es posible modificarla.

```
let string = 'Hola';
string[0] = 'h';
console.log(string[0]);
```

El ejemplo de arriba da como resultado un error. Para resolver esto, se puede crear una nueva cadena de texto y asignarla a la variable string reemplazando el string completo. Por ejemplo:

```
let string = 'Hola';
string = 'h' + string[1] + string[2] + string[3];
console.log(string);
```

Tipo Boolean

Este tipo se utiliza para almacenar valores de verdadero o falso. Generalmente se utilizan para probar una condición, después de lo cual se ejecuta el código. Los dos valores que pueden tener las variables booleanas son *true* o *false*, por ejemplo:

```
let test = 6 > 3;
console.log(test);
```

Tipo Undefined

El tipo *undefined* corresponde a las variables que han sido definidas y todavía no se les ha asignado un valor:

```
let edad;
```

Si una variable es declarada, pero no asignada, entonces su valor es *undefined*.

Tipo BigInt

El tipo BigInt se agregó recientemente al lenguaje JavaScript para representar números enteros de longitud arbitraria.

Un valor *BigInt* se crea agregando una "n" al final de un entero, por ejemplo:

```
const x = 12345678901234567890123456789012345678901234567890n;
```

Los BigInt no se pueden utilizar indistintamente con los *Number*, pues obtendremos un *TypeError*.

Tipo Symbol

El tipo *symbol* se utiliza para crear identificadores únicos para los objetos. Para crear un nuevo Símbolo primitivo, se escribe Symbol() con una cadena opcional como descripción:

```
let sym1 = Symbol();
let sym2 = Symbol('foo');
let sym3 = Symbol('foo');
```

Tipo Null

El tipo *null* se suele utilizar para representar objetos que no existen.

```
let edad = null;
```

El código anterior indica que el valor de la variable *edad* es desconocido o nulo.

Object

En JavaScript, los objetos son estructuras de datos complejas que tiene propiedades y métodos. Las propiedades expresan una cualidad del objeto, mientras que Los métodos son funcionalidades, es decir funciones o modos de operar asociados a un objeto.

Los objetos se crean mediante la palabra reservada *new* y el nombre de la clase que se va a instanciar, por ejemplo new Object, new Array, new Date, etc. También se pueden crear utilizando la notación literal, que una forma abreviada para crear objetos (u otros tipos de datos que veremos más adelante), sin necesidad de utilizar la palabra *new, por ejemplo:*

```
let perro = { name : 'Tobey', breed : 'Dalmatian' }; // Notación literal
```

Colecciones indexadas: arreglos

Los arreglos son objetos similares a una lista. Se representan mediante corchetes.

```
let frutas = ["Manzanas", "Peras", "Uvas", "Bananas", "Naranjas"];
console.log(frutas);
```

Date

Este objeto almacena la fecha, la hora, y proporciona los métodos para administrarlas. El objeto Date contiene un Number que representa los milisegundos transcurridos desde el 1 de Enero de 1970 UTC.

```
let now = new Date();
console.log( now ); // muestra en pantalla la fecha y la hora actuales;
```

Colecciones con clave: map, set, WeakMaps, WeakSets

Set y WeakSet representan un conjunto de objetos, mientras que Map y WeakMap se asocian un valor a un objeto. Estas estructuras de datos fueron introducidas con la especificación ECMAScript 6. Además, existen otros objetos especiales, tales como Math, Regexp, Promise, entre otros que estudiaremos en este curso.

Function

Una función es un fragmento de código que puede ser llamado por otro código o por sí mismo, o por una variable que haga referencia a la función.

```
function sumar(a, b) {
    return a + b;
}
let resultado = sumar(15, 15);
```

Envolturas de objetos primitivos en JavaScript

Con excepción de *null* y *undefined*, todos los datos primitivos tienen objetos equivalentes:

- **String** para el string primitivo.
- **Number** para el number primitivo.
- **BigInt** para el bigint primitivo.
- **Boolean** para el boolean primitivo.
- **Symbol** para el symbol primitivo.

Todos estos tipos de datos los conoceremos de forma detallada a medida que avancemos en este curso.