Módulo 2 - Variables y Tipos de Datos - Teoría

Contenido:

1. Sintaxis de JavaScript

Como todo lenguaje, existe una sintaxis que nos indica mediante un conjunto de reglas, cómo se construyen las declaraciones y finalmente, los programas de JavaScript.

```
var name, lastName, fullName; // Declarando variables

name = "Pedro";
lastName = "Martínez"; // Asignando valores

fullName = name + " " + lastName; // Calculando o procesando valores
```

Esta sintaxis nos indica, por ejemplo, cómo debemos escribir valores de tipo `String`

```
var name = "Pedro"; // Usando dobles comillas
var name = 'Pedro'; // Usando comillas simples
```

O cómo debemos escribir los valores de tipo `Number`

```
var age = 34; // Valor sin decimales
var cost = 10.99; // Valor con decimales
```

1. Declaraciones

Las declaraciones o sentencias en JavaScript son cada una de las instrucciones que deben ejecutarse para construir un programa.

```
var name = "Valeria"; // Declaración 1
console.log(name); // Declaración 2
name = "Óscar"; // Declaración 3
```

Están compuestas por valores, operadores, expresiones, palabras claves y comentarios; y serán ejecutadas en el mismo orden en el que fueron escritas (de arriba hacia abajo y de izquierda a derecha).

2. Variables

Las variables nos permiten almacenar valores de forma que puedan estar en memoria a la espera de ser usadas.

En JavaScript se declaran variables con la palabra clave `var` (también veremos más adelante que podemos declarar variables con las palabras `let` y `const`).

```
var total;
total = 10;
```

En nuestro caso, en la primera sentencia o declaración, hemos definido una variable llamada `total`. En la segunda sentencia, le estamos asignando el valor `10` mediante el signo de igualdad a la variable `total`.

El signo de igualdad `=` es usado para asignar un valor a la variable.

Declaración de variables

Podemos declarar nuestras variables en dos pasos:

```
var name;
name = "Nombre";
```

Y en un sólo paso:

```
var lastName = "Apellido";
```

Y también podemos declarar múltiples variables de una sola vez, separándolas por comas `,`:

```
var name, lastName, age;
```

```
var name = "Nombre",
  lastName = "Apellido",
  age = 34;
```

> Recuerda que cuando declaramos una variable pero no asignamos un valor, nos aparecerá con valor `undefined`.

3. Identificadores

Los nombres que usemos para definir nuestras variables se llaman `identificadores`.

Vamos a ver una serie de cuestiones a tener en cuenta sobre ellos:

* A la hora de nombrar nuestras variables debemos saber que **JavaScript es sensible al uso de mayúsculas y minúsculas**:

```
var total, Total;
total = 10;
Total = 30;
```

Si nos fijamos, son dos variables diferentes. Lo mismo ocurre cuando vamos a nombrar funciones o cuando usamos las palabras clave.

```
VAR name;   // -> Uncaught SyntaxError: Unexpected identifier
Var name;   // -> Uncaught SyntaxError: Unexpected identifier
var name;   // OK!
```

* Otra cosa a tener en cuenta es que **JavaScript no permite que tus variables o funciones comiencen con números**. El primer carácter debe ser una letra, un guión bajo (_) o un símbolo de dolar ($). El resto de caracteres pueden ser números, letras, guiones bajos o símbolos de dolar.

```
var name1;  // Permitido
var 1name;  // No permitido (Uncaught SyntaxError: Invalid or unexpected token)
var name_1, name$, _name, $name;  // Permitido
```

> Tampoco podemos nombrar a nuestras variables con las palabras reservadas.

4. Semicolons o puntos y comas `;`

Sirven para indicar el final de una declaración.

JavaScript permite omitir el uso de los *puntos y comas* porque tiene un sistema de inserción automática de puntos y comas, aunque debemos tener en cuenta una serie de reglas para poder omitirlos. Nosotros por simpleza y legibilidad, recomendamos usarlos, hasta el punto de configurar reglas en los linters (herramientas que nos ayudan a escribir nuestro código) para que avise de su ausencia.

Haciendo uso de los puntos y comas podemos escribir el siguiente código:

```
var name = "Valeria";
console.log(name);
name = "Óscar";
console.log(name);
```

De forma equivalente:

```
var name = "Valeria"; console.log(name); name = "Óscar"; console.log(name);
```

Si nos fijamos, haciendo uso de los puntos y comas podemos escribir múltiples declaraciones de JavaScript en una sola línea.

Ampliación de conceptos

El analizador de JavaScript agregará automáticamente un punto y coma cuando, durante el análisis del código fuente, encuentre estas situaciones:

1. la siguiente línea comienza con un código que rompe el actual (el código puede generar varias líneas).
2. la siguiente línea comienza con un `}`, cerrando el bloque actual.
3. se alcanza el final del archivo de código fuente.
4. hay una declaración `return` en su propia línea.
5. hay una declaración `break` en su propia línea.
6. hay una declaración `throw` en su propia línea.
7. hay una declaración `continue` en su propia línea.

5. Comentarios

Los comentarios sirven para indicar a JavaScript que esa sentencia no será ejecutada. Normalmente los usamos para "hablar" sobre el código, pero también para hacer pruebas sobre nuestro código comentando esas líneas que queremos omitir.

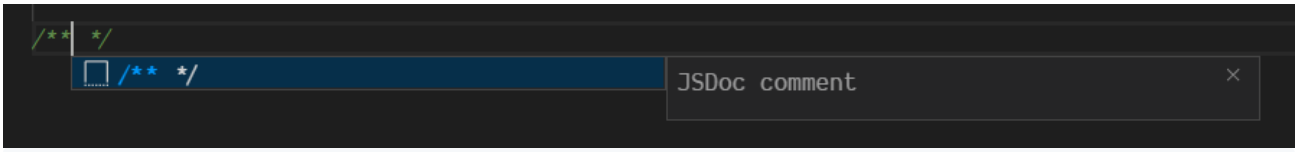
Tenemos comentarios de una sola línea o multilínea:

```
// comentario en una sola línea

/* comentario
  multilínea
 */

/* sin embargo no puedes /* anidar comentarios */ SyntaxError */
```

También puedes crearlos automáticamente en VSCode y con formato JSDoc según tecleas `/**`




```
/**
 * comentario multilínea con formato JSDoc
 * son muy útiles para documentar
 */
```

2. Tipos de datos

En JavaScript nuestras variables son tipadas de forma dinámica, lo que viene a decir que puedes asignar un texto a una variable y luego asignar un número... sin restricciones.

```
var age = "Edad"; // console.log(age) -> "Edad"
age = 34; // console.log(age) -> 34
```

Es importante saber con qué tipo de datos estamos trabajando para evitar errores a la larga.

Por ejemplo, debemos saber que si en una misma sentencia mezclamos números y texto, evaluará toda la sentencia como texto.

```
var datosPersonales = 34 + "edad"; // "34edad"
datosPersonales = "edad" + 34; // "edad34"
```

También, como dijimos anteriormente, debemos tener en cuenta que JavaScript ejecuta el código según escribimos, porque el orden en el que están escritas las sentencias importa.

```
var age = 3 + 4 + "edad";
```

```
var age = "edad" + 3 + 4;
```

1. typeof

Una forma de saber de qué tipo es una variable o simplemente si está declarada, es usando el operador `typeof`. Cuando lo usamos seguido de la variable que queremos consultar, nos responderá con el tipo al que pertenece.

```
var numero = 2;
console.log(typeof numero); // number

var texto = "MiTexto";
console.log(typeof texto); // string

var booleano = true;
console.log(typeof booleano); // boolean

console.log(typeof noExistoAun); // undefined
```

Vamos a comenzar con los tipos de datos esenciales en JavaScript:

2. Numbers

El tipo `number` representa cualquier número, ya sea un entero, un número negativo, en notación científica, etc.

```
var count = 5,
  diff = -4,
  percent = 4.7,
  scientific = 2e2,
  hex = 0x3;
```

3. Strings

Un `string` o cadena de caracteres se utiliza para almacenar una serie de caracteres como la siguiente cadena "Hola mundo". Un carácter es cada uno de los elementos de los cuales está formada la cadena.

```
var text1 = "Hola mundo";
var text2 = "Hola mundo";
```

Importante, si escribimos `var text = "25"` sigue siendo un `string` y no un `number`.

```
var text3 = "25"; // typeof text3 -> string
```

También debemos hacer hincapié en que si queremos que nuestro texto aparezca con comillas sin alternar entre dobles comillas y comillas simple, debemos "escaparlas"

```
var text4 = "Mi texto tiene \"comillas\"";
```

Esto es equivalente a

```
var text4 = 'Mi texto tiene "comillas"';
```

Podemos añadir una cadena al final de otra. Este proceso se llama **concatenar** y se realiza con el operador `+`

```
var name = "Laura",
  lastName = "Martínez";
alert("Hola " + name + " " + lastName);
```

4. Booleans

Los tipos `boolean`, en pocas palabras, sirven para permitir dos estados `true` o `false`.

```
var isTrue = true;
var isFalse = false;
```

5. Undefined

Como hemos dicho anteriormente, una variable cuando no existe o no tiene valor, digamos que tiene el valor `undefined`. Si consultáramos su tipo mediante `typeof` nos diría que es `undefined`.

```
var sinAsignar;
console.log(sinAsignar); // su valor es undefined
typeof sinAsignar; // tipo undefined
```

Podemos asignar el valor `undefined` a una variable.

```
var name = "Mi nombre";
console.log(name); // "Mi nombre"

name = undefined;
console.log(name); // undefined
```

6. Objects

Y ahora que hemos visto los tipos de datos esenciales de JavaScript (también llamados primitivos), ahora veremos como introducción el tipo de datos `object`.

Digamos que en JavaScript los `object` son como una colección de propiedades. Se crean con `{}` y dentro de ellas pondremos parejas de `clave: valor`, separadas por comas.

```
var coche = {
  marca: "Seat",
  modelo: "León",
  matricula: "0123BCD",
  itvCaducidad: 2020,
};

console.log(coche); // -> { marca: "Seat", modelo: "León", matricula: "0123BCD", itvCaducidad: 2020 }
```

Para acceder a una propiedad de nuestro objeto podemos hacerlo de dos formas:

```
coche.marca; // nombreObjeto.nombrePropiedad

coche["marca"]; // nombreObjeto["nombrePropiedad"]
```

Y para editar una propiedad, accedemos de la misma forma:

```
coche.modelo = "Ibiza"; // nombreObjeto.nombrePropiedad

coche["modelo"] = "Ibiza"; // nombreObjeto["nombrePropiedad"]

console.log(coche); // -> { marca: "Seat", modelo: "Ibiza", matricula: "0123BCD", itvCaducidad: 2020 }
```

Lo interesante de acceder a un objeto a través de la sintaxis de corchetes ([]) es que podemos pasarle una variable de tipo `string` con el identificador de la propiedad a la que queremos acceder. Es decir, imaginemos que tenemos un código en el que se "calcula" el nombre de la propiedad a la que queremos acceder en nuestro objeto coche. Ese nombre, como es calculado a partir de otras operaciones, digamos que es dinámico y lo guardamos en una variable.

```
var propiedadEnCoche = comprobamosAlgo ? "marca" : "modelo"; // alguna operación que devuelva true (y asignará "marc

console.log(coche[propiedadEnCoche]);
```

Imaginemos que nuestra comprobación resulta verdad y asigna `"marca"` a la variable `propiedadEnCoche`. Cuando la usemos en la siguiente sentencia del `console.log` será como haber hecho `coche["marca"]` porque el contenido de `propiedadEnCoche` es un `string` que contiene el valor `"marca"`.

En caso de que nuestra comprobación hubiera resultado falsa, se hubiese asignado `"modelo"` y utilizando `coche[propiedadEnCoche]` hubiéramos accedido a `coche["modelo"]`.

3. Interactuar con el usuario

alert

Ya hemos visto cómo podemos mostrar mensajes al usuario por pantalla a través de la función `alert()`. Recordemos que lo que hacíamos con esta función es pasar por parámetro un texto y nos lo mostraba a través del navegador por pantalla.

```
alert("Hola mundo"); // equivale a window.alert("Hola mundo")
```

prompt 

Tenemos una función que nos permite interactuar con el usuario y almacenar los valores que nos pasa. Es mediante el uso de la función `prompt()`.

```
var userName = prompt("Introduce un nombre de usuario");
alert(userName);
```

La función `prompt()` se utiliza como la función `alert()`, pero con una importante diferencia. Nos devuelve lo que el usuario haya introducido.

En caso de que haya cancelado la operación (también mediante la tecla de `Esc`) nos devolverá `null`.

confirm

También tenemos otra forma de comunicarnos con el usuario mediante el uso de la función `confirm()`. Es muy parecida a `prompt()` aunque ahora mostrará un cuadro de diálogo en el que el usuario sólo podrá responder Aceptar o Cancelar, y nos devolverá un booleano con el resultado. Es decir, devolverá `true` si ha elegido Aceptar y `false` en caso contrario.

Si el usuario cancela mediante `Esc` también devolverá `false`.

```
var aceptaCondiciones = confirm("¿Acepta las condiciones de contratación?");
alert(aceptaCondiciones);
```

> Hay que tener en cuenta que estas tres funciones detienen la ejecución del script hasta que el usuario haya interactuado con ellas.

Conclusiones

- Una variable es una manera de almacenar un valor en memoria.
- Usamos la palabra clave `var` para declarar una variable, y usamos `=` para asignar un valor a la variable.
- Las variables se escriben de forma dinámica, lo que significa que no es necesario especificar el tipo de contenido que la variable va a contener.
- El operador `+` concatena cadenas de caracteres, es decir, de inicio a fin.
- El operador `typeof` nos indica el tipo de datos de la variable consultada.
- Las funciones `prompt()` y `confirm()` permiten interactuar con el usuario.