

{JS}

Clase 02

JS
CONQUER BLOCKS

<índice>

Conceptos básicos del lenguaje

Sentencias, Bloques e Identificadores

Comentarios

Tipos de datos básicos y typeof

Declaración e inicialización

Sentencias, Bloques e identificadores

Identificadores

Identificadores

En JavaScript, los identificadores son nombres utilizados para identificar variables, funciones, clases y otros elementos en el código. Aquí hay algunas reglas y convenciones importantes sobre los identificadores en JavaScript

Identificadores

Reglas Básicas

- Deben comenzar con una letra (A-Z, a-z), un guion bajo (_) o un signo de dólar (\$).
- Los caracteres siguientes pueden ser letras, dígitos (0-9), guiones bajos o signos de dólar.
- No se permiten caracteres especiales o espacios.

Identificadores

Sensibilidad a Mayúsculas y Minúsculas:

- JavaScript es sensible a mayúsculas y minúsculas. Por ejemplo, variable, Variable, y VARIABLE son tres identificadores diferentes.

Identificadores

Palabras Reservadas:

- Hay ciertas palabras que no se pueden usar como identificadores porque están reservadas por el lenguaje, como if, for, let, const, etc.

Identificadores

Estilo de Nomenclatura:

- Camel Case: Usado comúnmente para nombrar variables y funciones (ejemplo: miVariable, calcularEdad).
- Pascal Case: A menudo utilizado para clases (ejemplo: Producto, CocheElectric).
- Snake Case: Menos común en JavaScript, pero usado a veces para constantes (ejemplo: MAX_VALOR, URL_BASE).

Identificadores

Identificadores Descriptivos:

- Se recomienda usar nombres descriptivos para hacer el código más legible (ejemplo: sumaTotal, nombreUsuario en lugar de s, n).

Identificadores

Identificadores Largos:

- No hay un límite de longitud específico para los identificadores, pero es una buena práctica mantenerlos de una longitud manejable.

Sentencias

Sentencias

En JavaScript, una sentencia es la unidad mínima de ejecución. Es una instrucción que le dice al navegador qué hacer. Cada sentencia en JavaScript puede llevar a cabo una acción, como declarar una variable, asignar un valor, ejecutar una función, controlar el flujo del programa mediante bucles y condicionales, y más.

Sentencias

Estructura básica

Una sentencia puede ser tan simple como una única palabra clave o variable, o tan compleja como una serie de llamadas de funciones y operaciones.

Terminación

Por lo general, las sentencias se terminan con un punto y coma (;). Sin embargo, debido a la Inserción Automática de Punto y Coma (ASI) en JavaScript, no siempre es necesario, aunque a menudo se recomienda para evitar errores sutiles.

Ejemplos

- Declaración de Variables: `let x = 5;`
- Asignación: `x = 10;`
- Funciones: `function miFuncion() { ... }`
- Condicionales: `if (x > 5) { ... }`
- Bucles: `for (let i = 0; i < 10; i++) { ... }`
- Expresiones: `console.log(x);`

Bloque

Bloque

En JavaScript, un bloque de código es una sección del código fuente que está delimitada por llaves `{ }`. Estos bloques definen un ámbito o contexto en el cual se agrupan múltiples sentencias de JavaScript. Los bloques de código son **fundamentales** en la estructura y organización del código en JavaScript, y son utilizados en varios contextos

Bloque

En JavaScript, un bloque de código es una sección del código fuente que está delimitada por llaves `{ }`. Estos bloques definen un ámbito o contexto en el cual se agrupan múltiples sentencias de JavaScript. Los bloques de código son **fundamentales** en la estructura y organización del código en JavaScript, y son utilizados en varios contextos

Bloque

Agrupación

Permiten agrupar varias sentencias juntas para que puedan ser tratadas como una sola unidad.

Esto es especialmente útil en estructuras de control de flujo como condicionales (if, else) y bucles (for, while).

Bloque

Ámbito de variables (Scope)

Las variables declaradas dentro de un bloque de código tienen un ámbito local. Esto significa que están accesibles únicamente dentro de ese bloque. Por ejemplo, una variable declarada dentro de un bloque if no es accesible fuera de ese bloque.

Bloque

Estructuras de control

En estructuras de control de flujo, los bloques de código definen las sentencias que se ejecutan en condiciones específicas (como en if, else) o en cada iteración de un bucle (for, while).

Bloque

Funciones

El ejemplo más claro de bloque de código

Bloque

```
if (condicion) {  
    // Bloque de código para 'if'  
    console.log("Condición cumplida");  
}
```

```
for (let i = 0; i < 10; i++) {  
    // Bloque de código para 'for'  
    console.log(i);  
}
```

```
function miFuncion() {  
    // Bloque de código para la función  
    console.log("Hola, mundo!");  
}
```

Bloque

Conclusión

Los bloques de código son una herramienta fundamental en JavaScript para organizar y controlar el flujo de ejecución del programa

Comentarios

Comentarios

En JavaScript, los comentarios son fragmentos de texto dentro del código fuente que son ignorados por el intérprete de JavaScript

Existen dos tipos de comentarios en JavaScript
Son útiles para explicar qué hace el código,
hacer anotaciones, o temporalmente
deshabilitar ciertas partes del código durante el
desarrollo y la depuración

Comentarios

Existen dos tipos de comentarios en JavaScript

Comentarios de una sola línea

Comentarios en varias líneas

Comentarios

Comentarios de línea

Comienzan con `//`. Todo el texto que sigue a `//` en esa línea es tratado como un comentario

```
// Esto es un comentario de una sola línea  
var x = 5; // Este también es un comentario, después de la sentencia
```

Comentarios

Comentarios de bloque

Comienzan con /* y terminan con */. Todo el texto entre /* y */ es tratado como un comentario, independientemente de cuántas líneas abarque.

```
/* Este es un comentario  
que abarca varias  
líneas */  
  
var y = 10;
```

Comentarios

Los comentarios de varias líneas son especialmente útiles para comentar secciones más grandes de código o para añadir descripciones más detalladas. Además, permiten "comentar" temporalmente grandes bloques de código que no se desean ejecutar durante ciertas fases del desarrollo.

Comentarios

Además se pueden utilizar para linters y
programas de post-procesado de Javascript

Comentarios

Un programador pasa el 80% del tiempo leyendo y el 20% escribiendo, no lo olvides

Tipos de datos básicos

Tipos de datos básicos

Tipos de datos básicos

- string
- number
- boolean
- undefined
- null
- bigint
- symbol

Tipos de datos básicos

String

Utilizado para representar datos textuales. Se compone de una secuencia de caracteres y puede ser definido utilizando comillas simples, dobles o acentos graves (para plantillas literales).

Ejemplo: var nombre = "Alice";

Comillas dobles, simple y backticks

Lo estudiaremos en profundidad

Tipos de datos básicos

Number

Representa tanto enteros como números de punto flotante.

JavaScript utiliza una representación de punto flotante de doble precisión para todos sus números.

Ejemplo: var edad = 25;

Lo estudiaremos en profundidad

Tipos de datos básicos

Boolean

Este tipo tiene dos valores posibles: true (verdadero) y false (falso). Es útil en operaciones lógicas y toma de decisiones en el flujo de control.

Ejemplo: var esMayorDeEdad = true;

Lo estudiaremos en profundidad

Tipos de datos básicos

Undefined

Se utiliza para indicar una variable que ha sido declarada pero aún no se le ha asignado un valor.

Ejemplo: var esMayorDeEdad;

Tipos de datos básicos

Undefined

Se produce cuando:

- Al declarar una variable pero no asignarle un valor.
- Al intentar acceder a una propiedad que no existe en un objeto.
- Al intentar acceder a un elemento que no existe en un arreglo.
- Cuando una función no tiene una sentencia return explícita, devuelve undefined.

Tipos de datos básicos

Null

Es un tipo que tiene un único valor: null.

Se utiliza para representar la ausencia intencional de un valor de objeto. En el campo de la programación el valor null siempre hace referencia a una dirección inválida

Ejemplo: var esMayorDeEdad = null;

Tipos de datos básicos

A los programadores principiantes de JavaScript les cuesta entender al diferenciar entre `undefined` y `null`, puesto que ambos tipos de datos significan ausencia de valor. No obstante, existen matices que los convierten en tipos de datos completamente distintos.

Tipos de datos básicos

Este es el más importante: undefined significa que no hay valor porque aún no se ha definido; en cambio, null significa que no hay valor porque así lo ha indicado expresamente el programador.

Tipos de datos básicos

Entender la diferencia entre `undefined` y `null` es crucial para manejar adecuadamente los estados de las variables y los errores en JavaScript.

Tipos de datos básicos

BigInt

Introducido en versiones recientes de JavaScript, este tipo permite trabajar con números enteros muy grandes que superan el límite de los números del tipo Number.

Ejemplo: var numeroGrande = 1234567890123458901234567890n;

Tipos de datos básicos

Symbol

Symbol: Introducido en ECMAScript 2015, es un tipo de datos cuyas instancias son únicas e inmutables. Son útiles para crear identificadores únicos para propiedades de objetos.

Ejemplo: let miSímbolo = Symbol('mi identificador único');

Tipos de datos básicos

Demos

Tipos de datos básicos

¿Qué tipo de dato tiene
una variable?

Tipos de datos básicos

`typeof`

es utilizado para determinar el tipo de una variable o expresión. Esto es especialmente útil en JavaScript debido a su naturaleza de tipado dinámico, donde el tipo de una variable puede cambiar en tiempo de ejecución.

Typeof

typeof

- Sintaxis: typeof operando, donde operando es la variable o expresión cuyo tipo se quiere conocer.
- Retorno: Devuelve una cadena de texto que indica el tipo del operando.

Typeof

typeof

Valores posibles de retorno:

`undefined`, `boolean`, `number`, `string`, `symbol`,
`object` o `function`.

Typeof

importancia de typeof

1. Depuración y Validación: Permite a los desarrolladores verificar el tipo de las variables durante la depuración y validar tipos de datos antes de realizar operaciones específicas.
2. Evitar Errores: Al verificar el tipo de una variable antes de operar con ella, se pueden prevenir errores comunes como intentar realizar operaciones inadecuadas para un tipo de dato específico.

Typeof

limitaciones de typeof

- No puede diferenciar entre un objeto y un arreglo, o entre un objeto y null. Para estos casos, se suelen utilizar otros métodos como `Array.isArray()` para arreglos o comparaciones directas con null.
- No puede identificar tipos de objetos más específicos (como instancias de clases personalizadas). Para eso usaremos `instanceof` más adelante

Typeof

Nos quedan por ver los tipos
no primitivos

Declaración e inicialización

Declaración e inicialización

En JavaScript, existen principalmente tres formas de declarar variables, cada una con sus propias características y niveles de alcance (scope).

Estas son var, let, y const.

¿Os acordáis de lo que era un bloque de código?

Declaración e inicialización

var

- var es la forma más antigua de declarar variables en JavaScript.
- Tiene un alcance (scope) de función; es decir, una variable declarada con var está disponible en toda la función en la que fue declarada, o globalmente si se declara fuera de una función.
- Permite la redeclaración de la misma variable en el mismo ámbito.
- Tiene lo que se conoce como "hoisting" (elevación), lo que significa que se puede usar la variable antes de su declaración en el código.

Declaración e inicialización

let

- Introducido en ECMAScript 2015 (ES6), let permite declarar variables con alcance de bloque (block scope), lo que significa que la variable solo existe dentro del bloque en el que fue declarada.
- No permite la redeclaración de la misma variable dentro del mismo bloque.
- Menos propenso a errores que var debido a su alcance de bloque y no permite el uso de la variable antes de su declaración.

Declaración e inicialización

const

- También introducido en ES6, const se utiliza para declarar constantes.
- Tiene un alcance de bloque, similar a let.
- No permite la redeclaración ni la reasignación de la variable.
- Debe ser inicializada en el momento de su declaración.
- Aunque una variable declarada con const es inmutable en términos de reasignación de su valor primitivo, si se trata de un objeto, las propiedades de ese objeto pueden ser modificadas.

Declaración e inicialización

var, let o const

- var se usa cada vez menos en el desarrollo moderno de JavaScript debido a sus peculiaridades y potenciales problemas con el alcance de función y hoisting.
- let es preferible cuando se necesita una variable cuyo valor va a cambiar, como contadores en bucles, o valores que se reasignan en un bloque de código.
- const es la mejor opción para declarar variables que no deben cambiar después de su asignación inicial. Esto mejora la legibilidad del código y reduce la posibilidad de errores inesperados.

Anatomía de una variable

Mi consejo, no uses var salvo que quieras variables globales y entiendas bien cómo funcionan

Anatomía de una variable

¿Qué podemos guardar en las variables?

Ejercicios

Ejercicios de tipos primitivos

1. Declara una variable saludo y asignale el texto "Hola Mundo".
2. Declara una variable edad y asignale tu edad.
3. Declara una variable estaSoleado y asigna un valor booleano dependiendo si está soleado o no.
4. Declara una variable valorNulo y asignale el valor null.
5. Declara una variable sinDefinir sin asignarle un valor.
6. Declara una variable numeroGrande y asignale un número entero grande usando BigInt.
7. Declara una variable mensaje y asignale el texto 'Aprendiendo JavaScript' usando comillas simples.
8. Declara una variable precio y asignale un valor decimal, por ejemplo, el precio de un artículo.
9. Declara una variable estaLloviendo y asigna el valor opuesto a estaSoleado.
10. Declara una variable temperatura y asignale un número negativo para representar una temperatura bajo cero.

<Despedida>

Email

bienvenidosaez@gmail.com

Instagram

@bienvenidosaez

Youtube

youtube.com/bienvenidosaez

CONQUERBLOCKS