4.2. Variables, tipos de datos y operadores

LENGUAJES DE PROGRAMACIÓN NIVEL 3. UNIDAD 4

JavaScript - LENGUAJE DE PROGRAMACIÓN PARA LA INTERACCIÓN EN EL FRONTEND

Financiado por la Unión Europea

Plan de Recuperación, Transformación y Resiliencia

Contenido

Introducción

Tras haber explorado la sintaxis básica de JavaScript, este módulo se adentra en tres pilares fundamentales para cualquier desarrollador: las variables, los tipos de datos y los operadores. Estos conceptos no solo se construyen sobre la base de la sintaxis que ya has aprendido, sino que también son esenciales para desarrollar scripts dinámicos y funcionales en JavaScript. Entender cómo almacenar, manipular y operar con datos es crucial para implementar lógica en tus programas y hacer que tus páginas web sean interactivas y reactivas a la entrada del usuario.

Las variables son como contenedores que almacenan datos que pueden ser utilizados y modificados a lo largo de tu programa. Aprenderás a declarar variables y a entender el alcance y la vida útil de estos datos dentro de tus scripts. La correcta utilización y nombramiento de las variables son prácticas clave para escribir código claro y mantenible.

Por otro lado, JavaScript es un lenguaje con una variedad rica de tipos de datos, desde simples como números y cadenas hasta más complejos como objetos y arrays. Comprender los diferentes tipos de datos y cómo interactúan es fundamental para manejar la información de manera efectiva y realizar operaciones sobre ellos.

Finalmente, los operadores te permiten realizar acciones sobre tus variables y valores. Desde simples operaciones matemáticas hasta comparaciones complejas, los operadores son las herramientas que te permiten implementar la lógica en tus programas. Aprenderás cómo estos pueden ser utilizados para tomar decisiones y controlar el flujo de ejecución de tus scripts.

Este módulo es un paso crucial en tu viaje de aprendizaje de JavaScript. Al conocer las variables, los tipos de datos y los operadores, estarás bien equipado para abordar desafíos más avanzados y crear experiencias web ricas e interactivas.

Variables

Las variables son uno de los conceptos más fundamentales en cualquier lenguaje de programación, actuando como "contenedores" para almacenar datos que se pueden utilizar y modificar a lo largo de tu programa.

En JavaScript, la comprensión profunda de cómo declarar, asignar y utilizar variables es crucial para controlar y manipular eficazmente los datos dentro de tus scripts.

Declaración y asignación de variables

Declarar una variable en programación significa reservar espacio en la memoria para almacenar un valor y asociar un nombre identificador con ese espacio.

En JavaScript, las variables se pueden declarar utilizando let, const, y (aunque menos recomendado debido a sus peculiaridades) var.

Examinemos una a una esas formas de declaración:

let:

Permite declarar variables cuyo valor puede cambiar.

Las variables declaradas con let tienen un alcance de bloque, lo que significa que existen dentro del bloque en el que se declaran y no son accesibles fuera de él. Es decir, su alcance es limitado al bloque en el que están declaradas.

```
let edad = 25;
edad = 26; // El valor puede ser actualizado
```

[1]

Ejemplo:

```
let contador = 0; // Declaración de una variable con `let`
contador = contador + 1; // Actualización del valor de la variable
console.log(contador); // Imprime 1 en la consola
```

[2]

En este ejemplo, contador es una variable que probablemente cambiará su valor a lo largo del tiempo, como sucede en bucles o cuando se cuenta la cantidad de veces que ocurre un evento. El uso de let es apropiado aquí porque permite la reasignación de valores.

[1] Accesibilidad:
let edad = 25;
edad = 26; //El valor de una variable puede ser actualizado

[2] Accesibilidad:
let contador = 0; // Declaración de una variable con let
contador = contador + 1 // Actualización del valor de una variable
console.log(contador); //Imprime 1 en la consola

const:

Se utiliza para declarar constantes, cuyo valor no se espera que cambie.

Al igual que let, const también tiene un alcance de bloque.

```
const NOMBRE = "Alice";
// NOMBRE = "Bob"; // Esto provocaría un error, ya que no se puede reasignar una con
```

[3]

Ejemplo

```
const PI = 3.14159; // Declaración de una constante con `const`
console.log(PI); // Imprime 3.14159 en la consola
```

[4]

Aquí, PI es una constante que representa el valor de Pi. Dado que Pi es un valor matemático que no cambia, se utiliza const para declararlo, asegurando que el valor permanezca constante y no pueda ser reasignado accidentalmente en otra parte del código

var

Su uso no recomendado en contextos modernos.

Antes de ES6[5], var era la única forma de declarar variables en JavaScript. Las variables declaradas con var tienen un alcance de función o global, lo que puede llevar a comportamientos inesperados en programas complejos.

[3] Accesibilidad:
const NOMBRE = “Alice”;
// NOMBRE = Bob; // Esto provocaría un error, ya que no se puede reasignar una constante

[4] Accesibilidad:
const PI = 3.14159; // Declaración de una constante con “const”
console.log(PI); // Imprime 3.14159 en la consola

[5] ES6, también conocido como ECMAScript 2015, es una versión significativa del estándar ECMAScript que fue publicada en junio de 2015. ECMAScript es el estándar en el que se basa JavaScript, y cada nueva versión de ECMAScript trae consigo nuevas características, mejoras en la sintaxis, y mejores prácticas para hacer el lenguaje de programación más poderoso y fácil de usar.

Reglas y convenciones para nombrar variables

Las más relevantes son las siguientes:

- Case Sensitivity: JavaScript es sensible a mayúsculas y minúsculas. Por lo tanto, variable, Variable, y VARIABLE serían identificadores diferentes.
- Estilo camelCase: Es común utilizar camelCase para nombrar variables en JavaScript, donde la primera palabra está en minúsculas y cada palabra subsiguiente comienza con una mayúscula.

```
let miVariableImportante;
```

[6]

- Evitar palabras reservadas: No debes usar palabras reservadas por JavaScript (como let, const, return, etc.) como nombres de tus variables.
- Nombres descriptivos: Elige nombres que describan claramente el propósito o el contenido de la variable, lo que hace que tu código sea más legible.

[6] Accesibilidad:
let miVariableImportante;

Tipos de datos

Después de haber explorado las variables y cómo se utilizan para almacenar información en JavaScript, es crucial entender los diferentes tipos de datos que puedes asignar a estas variables.

Los tipos de datos en JavaScript se dividen en dos categorías principales: primitivos y complejos. Comprender estos tipos es fundamental para manipular datos de manera efectiva en tus programas, realizar operaciones y aplicar lógica condicional.

Tipos de datos primitivos

Los tipos de datos primitivos representan valores simples y son inmutables, lo que significa que no puedes cambiar el valor de un dato primitivo una vez que se crea. JavaScript define varios tipos primitivos:

String:

Representa secuencias de caracteres y se utiliza para trabajar con texto. Se pueden definir usando comillas simples, dobles o comillas invertidas.

```
let saludo = "Hola, mundo!";
```

7

Number:

Incluye tanto números enteros como decimales. JavaScript utiliza un único tipo Number para todos los números, independientemente de si son enteros o decimales.

```
let edad = 25;
let precio = 99.99;
```

8

[7] Accesibilidad:
let saludo = “Hola, mundo!”;

[8] Accesibilidad:
let edad =25;
let precio = 99.99;

Boolean:

Tiene solo dos valores posibles, true o false. Es útil para tomar decisiones lógicas o controlar estructuras de control como bucles e instrucciones condicionales.

```
let esMayorDeEdad = true;
```

9

undefined:

Indica que una variable ha sido declarada pero aún no se le ha asignado un valor.

null:

Representa la ausencia intencional de cualquier valor de objeto. Se utiliza para indicar que una variable no apunta a ningún objeto o valor específico.

[9] Accesibilidad:
let esMayorDeEdad = true;

Tipos de datos complejos

En JavaScript, cuando hablamos de tipos de datos complejos, nos referimos principalmente a objetos.

A diferencia de los tipos de datos primitivos (como String, Number, Boolean, null, y undefined), que almacenan un único valor, los objetos pueden contener una colección compleja de datos y/o funcionalidades.

Objetos

Los objetos en JavaScript son básicamente colecciones de propiedades, donde cada propiedad es un par de clave-valor.

Estas claves son cadenas de texto (o Symbols), y los valores pueden ser cualquier tipo de dato, incluyendo otros objetos, lo que permite la creación de estructuras de datos complejas.

```
let persona = {
    nombre: "Alice",
    edad: 30,
    esEstudiante: true
};
```

10

En este ejemplo, el objeto “persona” incluye una estructura de datos compuesta por tres pares clave-valor”.

Como ves, un objeto se puede crear utilizando llaves {}, con pares clave-valor separados por comas y en línea diferente. Cada clave-valor se separa por dos puntos.

[10] Accesibilidad

```
let persona = {
    nombre: “Alice”,
    edad = 30,
    esEstudiante = True
};
```

Dentro de la categoría de objetos, hay tipos especiales que incluyen:

- Array:
  En JavaScript, los arrays son técnicamente un tipo especial de objeto diseñado para almacenar colecciones de datos. Aunque los arrays comparten muchas características con los objetos generales, como ser colecciones de propiedades, están optimizados para representar y manejar conjuntos ordenados de datos y vienen con una serie de métodos y propiedades específicos para trabajar con esos datos de manera secuencial.

```
let frutas = ["manzana", "banana", "cereza"];
console.log(frutas[0]); // Accede al primer elemento, "manzana"
frutas.push("naranja"); // Agrega "naranja" al final del array
```

[11]

Como ves, un array se escribe como una lista de valores entre corchetes [], separados por comas.

[11] Accesibilidad:
let frutas = [“manzana”, “banana”, “cereza”];
console.log(frutas[0]); // Accede al primer elemento “manzana”
frutas.push(“naranja”); // Agrega “naranja” al final del array

- Function:
  Son un tipo especial de objeto conocido como objetos de función. Esto significa que las funciones en JavaScript tienen propiedades y métodos, como cualquier otro objeto, y que pueden ser manipuladas y pasadas alrededor de tu código como cualquier otro valor de objeto. Esto significa que pueden ser almacenados en variables, pasados como argumentos a otras funciones y creados dinámicamente durante la ejecución del programa.

12

¿Cuándo usarlos?

La decisión de cuándo usar objetos, arrays y funciones en JavaScript depende de la naturaleza de los datos que estás manejando y de lo que necesitas hacer con ellos. Aquí te doy una guía general sobre cuándo es apropiado usar cada uno:

Objetos

Usa objetos cuando:

- Necesites representar entidades con propiedades nombradas: Los objetos son ideales para representar entidades del mundo real o conceptuales con características claras, como una persona con nombre, edad, y email, entre otros.
- Quieras agrupar datos y funciones relacionadas: Los objetos pueden contener tanto datos como funciones (métodos) que operan sobre esos datos, encapsulando así la funcionalidad.

[12] Accesibilidad:
function saludar() {
console.log(“Hola, mundo!”);
}

Arrays

Usa arrays cuando:

- Tengas una colección de elementos ordenados: Los arrays son útiles cuando necesitas mantener un conjunto ordenado de elementos, especialmente si planeas recorrerlos secuencialmente o acceder a ellos por su índice.
- Necesites realizar operaciones sobre listas de datos: Los arrays vienen con una serie de métodos integrados para realizar transformaciones, filtrado, ordenamiento, etc., lo que los hace muy útiles para trabajar con colecciones de datos.

Funciones

Usa funciones cuando:

- **Quieras reutilizar un bloque de código:** Las funciones te permiten encapsular un conjunto de instrucciones que realizan una tarea específica, de modo que puedas reutilizar ese bloque de código simplemente llamando a la función.
- **Necesites modularizar tu código:** Las funciones ayudan a mantener tu código organizado, dividiéndolo en pequeñas unidades lógicas. Esto hace que tu código sea más legible y fácil de mantener.
- **Desarrolles operaciones complejas que dependan de entradas variables:** Al definir funciones que aceptan parámetros, puedes diseñar operaciones que se comporten de manera diferente según los argumentos que se les pasen.
- **Implementes patrones de programación avanzados:** Como las funciones en JavaScript son objetos de primera clase, puedes usarlas para implementar cierres, funciones de orden superior, y otros patrones avanzados que añaden poderosas capacidades a tus programas.

En resumen, la elección entre objetos, arrays y funciones dependerá de la estructura de tus datos y de tus necesidades específicas de programación. A menudo, encontrarás que estos se usan en conjunto para construir aplicaciones complejas y eficientes.

Operadores

Los operadores en JavaScript son símbolos especiales o palabras clave que se utilizan para realizar operaciones sobre uno o más operandos y devolver un resultado. Desde realizar operaciones matemáticas básicas hasta comparar valores o trabajar con lógica booleana, los operadores juegan un papel crucial en la construcción de la lógica y la manipulación de datos en la programación.

Este módulo explora los distintos operadores disponibles en JavaScript, destacando cómo y cuándo utilizar cada uno para efectuar cálculos, tomar decisiones y manipular datos dentro del contexto de los tipos de datos, variables y estructuras de control que hemos visto anteriormente.

Tipos de operadores

Los más básicos y comunes son los siguientes:

Operadores aritméticos

- Realizan operaciones matemáticas comunes.
- Tipos: + Suma, - Resta, * Multiplicación, / División, % Módulo, ++ Incremento, -- Decremento.
- Ejemplo de uso: Calcular el total de una compra, aumentar un contador, etc.

Operadores de asignación

- Asignan valores a las variables.
- Tipos: = Asignación básica, += Suma y asignación, -= Resta y asignación, etc.

- Ejemplo de uso: Establecer el valor de una variable, actualizar el valor de una variable acumulando un nuevo valor, etc.

Operadores de comparación

- Comparan valores y devuelven un booleano.
- Tipos: == Igualdad, === Igualdad estricta, != Desigualdad, !== Desigualdad estricta, > Mayor que, < Menor que, etc.
- Ejemplo de uso: Verificar si dos valores son iguales, determinar el mayor de dos números, etc.

Operadores lógicos

- Se utilizan para combinar expresiones booleanas.
- Tipos: && AND, || OR, ! NOT.
- Ejemplo de uso: Evaluar múltiples condiciones para tomar una decisión, invertir un valor booleano, etc.

No obstante, existen otros operadores que permiten realizar operaciones más complejas como los Ternarios, los de Cadena, los de Tipo, etc... Si sigues profundizando en tu conocimiento de JavaScript los conocerás.

Conclusión

Al llegar al final de este módulo, hemos cubierto tres pilares fundamentales de la programación en JavaScript: las variables, los tipos de datos y los operadores. Estos elementos son esenciales para cualquier desarrollador que busque manipular datos de manera efectiva y expresar lógica dentro de sus programas. Las variables nos permiten almacenar y etiquetar datos, los tipos de datos definen la naturaleza de los datos que manipulamos, y los operadores nos dan las herramientas para realizar operaciones sobre esos datos.

Lo que hemos cubierto en este módulo sienta las bases para todo lo que sigue. A medida que continúes explorando JavaScript, te encontrarás volviendo una y otra vez a estos conceptos básicos, aplicándolos de nuevas maneras y combinándolos con estructuras y patrones más avanzados.

Gracias por acompañarme en este viaje. Espero que lo aprendido aquí te sirva como una sólida base para tu desarrollo como programador y te inspire a seguir explorando y profundizando en este versátil lenguaje de programación.

Nos vemos en la siguiente lección.

Recursos adicionales

Para reforzar y expandir tu comprensión de los temas tratados en este módulo, aquí tienes una selección de recursos adicionales. Estos enlaces te llevarán a documentación oficial y tutoriales en línea que te proporcionarán una visión más detallada de las variables, los tipos de datos y los operadores en JavaScript.

Documentación oficial

- MDN Web Docs - JavaScript: Una de las mejores fuentes de documentación para desarrolladores web. Ofrece explicaciones detalladas, ejemplos de código y buenas prácticas.
  https://developer.mozilla.org/en-US/docs/Web/JavaScript

Tutoriales y guías

- JavaScript.info: Un recurso moderno y completo que cubre JavaScript desde los fundamentos hasta temas avanzados, con ejemplos claros y prácticos.
  https://javascript.info/
- freeCodeCamp: Ofrece un extenso curso de JavaScript como parte de su currículum de desarrollo web, completo con ejercicios prácticos y proyectos.
  https://www.freecodecamp.org/

Plataformas interactivas

- Codecademy: Proporciona un curso interactivo de JavaScript que es excelente para principiantes absolutos, ayudándote a aprender mediante la práctica.
  https://www.codecademy.com/

Blogs y artículos

- W3Schools JavaScript Tutorial: Un recurso educativo gratuito que ofrece tutoriales sobre una amplia gama de tecnologías web, incluyendo JavaScript. Es conocido por sus tutoriales interactivos que permiten a los usuarios experimentar con el código directamente en el navegador.
  https://www.w3schools.com/js/

Estos recursos te proporcionarán una base sólida y te ayudarán a profundizar en cada uno de los temas tratados, mejorando tu comprensión y habilidad en la manipulación efectiva de datos en JavaScript. ***A medida que explores estos recursos, recuerda practicar regularmente y experimentar con código propio para consolidar tu aprendizaje.***