4.3. Estructuras de control: condicionales y bucles

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

En el vasto mundo de la programación, las estructuras de control son esenciales para dotar a nuestros programas de lógica y capacidad de decisión.

Sin ellas, nuestros códigos serían secuencias lineales sin flexibilidad ni interactividad. Las estructuras de control nos permiten dirigir el flujo de ejecución de los programas, tomando decisiones basadas en condiciones (a través de estructuras condicionales) y repitiendo operaciones de forma controlada (mediante estructuras iterativas o bucles).

Importancia de las estructuras de control

Las estructuras de control son fundamentales para cualquier lenguaje de programación, incluido JavaScript, ya que permiten que nuestros programas respondan de manera diferente según los datos de entrada y las condiciones dadas. Estas estructuras nos dan la capacidad de:

- Ejecutar diferentes bloques de código basados en condiciones específicas.
- Repetir un conjunto de operaciones múltiples veces sin duplicar el código.
- Iterar sobre estructuras de datos como arrays y objetos, permitiéndonos manipular y acceder a sus elementos de manera eficiente.

Relación con conceptos previos

Las estructuras de control en JavaScript se construyen sobre los fundamentos que ya has aprendido:

- **Variables:** Las estructuras de control a menudo utilizan variables para almacenar los datos que se evalúan en sus condiciones o que cambian con cada iteración de un bucle.
- **Tipos de Datos:** Comprender los diferentes tipos de datos es crucial para establecer condiciones significativas. Por ejemplo, saber cuándo

una operación devuelve un booleano es esencial para las expresiones condicionales.

- **Operadores:** Los operadores juegan un papel vital en las estructuras de control. Los operadores de comparación (==, !=, <, >, <=, >=) y los operadores lógicos (&&, ||, !) son especialmente importantes, ya que se utilizan para formular las condiciones bajo las cuales se ejecutan los bloques de código en las estructuras condicionales y los bucles.

Al combinar estos elementos, las estructuras de control te permiten crear programas dinámicos y reactivos que pueden tomar decisiones y repetir acciones basadas en la lógica y las condiciones definidas, lo que lleva tus habilidades de programación en JavaScript a un nuevo nivel.

Estructuras condicionales

Las estructuras condicionales son, sin duda, una de las estructuras de control fundamentales en cualquier lenguaje de programación, incluido JavaScript. Permiten que un programa tome decisiones y ejecute diferentes bloques de código según si ciertas condiciones son verdaderas o falsas. En este sentido, son herramientas esenciales para añadir lógica a tus programas.

Papel de las estructuras condicionales

Las estructuras condicionales son principalmente utilizadas para:

- **Tomar decisiones:** Basándose en el estado actual de los datos del programa, se puede decidir qué bloque de código ejecutar.
- **Validación de datos:** Comprobar si los datos de entrada cumplen con ciertos criterios antes de proceder con más operaciones.
- **Ramas en la lógica del programa:** Crear diferentes caminos o ramas en la lógica del programa, lo que permite una mayor flexibilidad y adaptabilidad a diferentes situaciones o entradas

La importancia de las estructuras condicionales

Al comprender y aplicar estructuras condicionales, estás equipando tus programas con la capacidad de reaccionar y adaptarse. Esto no solo mejora la interactividad, sino que también permite el manejo efectivo de la validación de datos, la lógica de negocios y el control de flujo dentro de tus aplicaciones.

Principales estructuras condicionales

Las principales son las siguientes:

Sentencia if

La sentencia if es la forma más básica de una estructura condicional. Ejecuta un bloque de código solo si una condición especificada es verdadera.

```
if (condición) {
    // Bloque de código que se ejecuta si la condición es verdadera
}
```

1

Sentencia else

La cláusula else acompaña a una sentencia if y ejecuta un bloque de código alternativo si la condición if es falsa.

```
if (condición) {
    // Bloque de código si la condición es verdadera
} else {
    // Bloque de código alternativo si la condición es falsa
}
```

2

[1] Accesibilidad:

```
if (condición) {
    // Bloque de código que se ejecuta si la condición es verdadera
}
```

[2] Accesibilidad:

```
if (condición) {
    // Bloque de código que se ejecuta si la condición es verdadera
} else {
    // Bloque de código alternativo si la condición es falsa
}
```

Sentencia else if

Para más complejidad y múltiples condiciones, puedes encadenar sentencias if con else if. Esto te permite probar varias condiciones de forma secuencial.

```
if (condición1) {
    // Bloque de código si condición1 es verdadera
} else if (condición2) {
    // Bloque de código si condición2 es verdadera
} else {
    // Bloque de código si ninguna de las condiciones anteriores es verdadera
}
```

3

Operador ternario

El operador ternario es una forma compacta de escribir una instrucción if-else. Es especialmente útil para asignaciones basadas en una condición.

```
condición ? expresión1 : expresión2;
```

4

Consta de tres partes:

- una condición, seguida de un signo de interrogación (?),
- luego la expresión a ejecutar si la condición es verdadera,
- seguida de dos puntos (:),
- y finalmente la expresión a ejecutar si la condición es falsa.

---

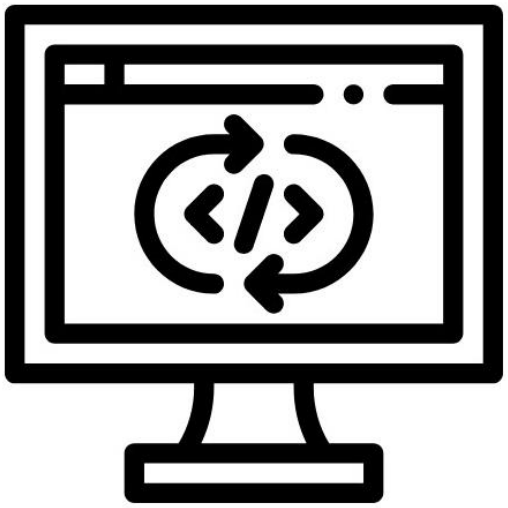
[3] Accesibilidad

```
if (condición1) {
    // Bloque de código que se ejecuta si la condición1 es verdadera
} else if (condicion2) {
    // Bloque de código que se ejecuta si la condicion2 es verdadera
} else {
    // Bloque de código si ninguna de las dos condiciones es verdadera
}
```

[4] Accesibilidad:

```
condición ? expresión1 : expresion2;
```

Estructuras iterativas (bucles)



Las estructuras iterativas, comúnmente conocidas como bucles, son herramientas indispensables en la programación que permiten repetir un bloque de código múltiples veces. Estas estructuras son esenciales para operar con conjuntos de datos, realizar tareas repetitivas y automatizar procesos dentro de tus programas.

En este módulo, exploraremos las diversas formas de bucles en JavaScript y cómo cada una se adapta a diferentes situaciones, construyendo sobre los conceptos fundamentales de variables, tipos de datos y operadores que ya hemos visto.

Tipos de bucles

Son los siguientes:

Bucle for

El bucle for es una de las formas más comunes de iteración en JavaScript, ideal para cuando sabes cuántas veces necesitas repetir un bloque de código

```
for (inicialización; condición; actualización) {
    // Código a ejecutar en cada iteración
}
```

5

El bucle for en JavaScript es una estructura de control que permite repetir un bloque de código un número determinado de veces. Como se ve en la imagen, su sintaxis consta de tres partes opcionales separadas por punto y coma dentro de paréntesis:

- la inicialización, donde generalmente se establece un contador;
- la condición, que es evaluada antes de cada iteración y determina si el bucle continuará ejecutándose;
- y la expresión final, que se ejecuta al final de cada iteración, comúnmente usada para actualizar el contador.

Estas tres partes están seguidas por el bloque de código a ejecutar, encerrado entre llaves.

[5] Accesibilidad:
for (inicialización; condición; actualización) {
// Código a ejecutar en cada iteración
}

Por ejemplo:

```
for (let i = 1; i <= 5; i++) {
    console.log(i);
}
```

6

Se inicia un contador i en 1, continúa el bucle mientras i sea menor o igual que 5, e incrementa i en 1 después de cada iteración, imprimiendo el valor de i en cada paso lo que hace que dse impriman en pantalla los números de 1 a 5.

Este bucle es especialmente útil para iterar sobre arrays y colecciones o cuando se conoce de antemano el número de repeticiones deseado.

Bucle while

El bucle while repite un bloque de código mientras una condición especificada sea verdadera, lo que lo hace adecuado para situaciones en las que el número de iteraciones no es conocido de antemano.

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

7

La sintaxis del bucle while comienza con la palabra clave while, seguida de la condición entre paréntesis, y luego el bloque de código a ejecutar, encerrado entre llaves.

Antes de cada iteración, se evalúa la condición: si resulta ser verdadera, el bloque de código se ejecuta, y luego la condición se vuelve a evaluar. Este

---

[6] Accesibilidad:

```
for (let i =1; i <= 5; i++) {
    console.log(i);
}
```

[7] Accesibilidad:

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

proceso continúa hasta que la condición se evalúa como falsa, momento en el cual el bucle termina y el control pasa a la siguiente instrucción después del bucle.

Es crucial que dentro del bloque de código del bucle while, haya alguna operación que eventualmente haga que la condición sea falsa; de lo contrario, el bucle se convertiría en un bucle infinito. El bucle while es particularmente útil cuando el número de iteraciones no se conoce antes de comenzar el bucle.

Por ejemplo:

```
let contador = 0;
while (contador < 3) {
    console.log("Hola");
    contador++;
}
```

[8]

Este ejemplo hace que se imprima en pantalla dos veces la palabra Hola. Implica que previamente se defina la variable “contador” para que posteriormente en el bloque While se defina la confición (hasta que el contados llegue a dos) que se debe ejecutar el Hola y el incremento de una unidad de la variable.

[8] Accesibilidad:

```
let contador = 0;
while (contador < 3) {
    console.log(“Hola!”);
    contador++;
}
```

Bucle do...while

El bucle do...while es similar al while, con la diferencia clave de que garantiza que el bloque de código se ejecute al menos una vez.

```
do {
    // Código a ejecutar
} while (condición);
```

[9]

Como hemos visto, en un bucle while, la condición se evalúa antes de ejecutar el bloque de código dentro del bucle. Esto significa que, si la condición inicial es falsa, el bloque de código dentro del while no se ejecutará ni una sola vez.

En un bucle do...while, el bloque de código se ejecuta una vez antes de que la condición sea evaluada por primera vez. Esto garantiza que el bloque de código dentro del bucle se ejecute al menos una vez, independientemente de la condición inicial.

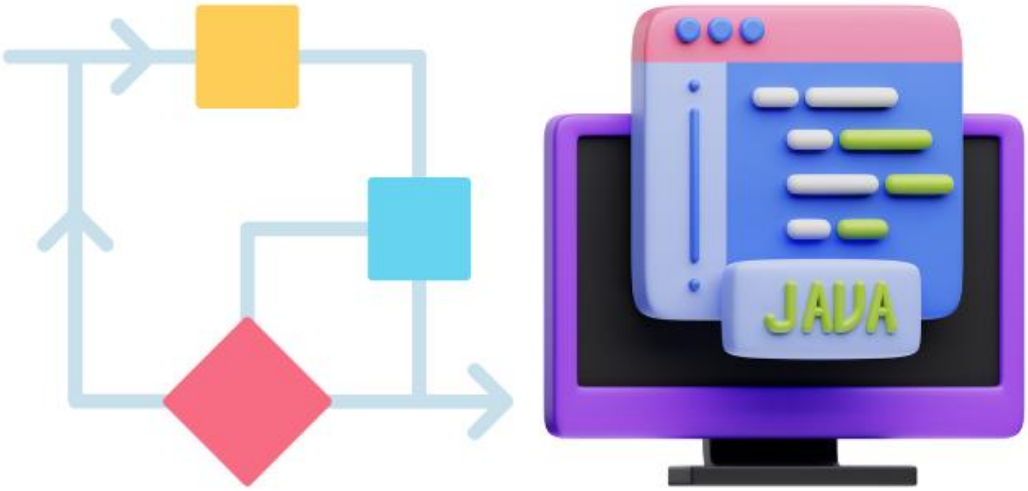
Otros bucles

Existen otros bucles como:

- for...in: Itera sobre todas las propiedades enumerables de un objeto.
- for...of (ES6): Itera sobre los valores de objetos iterables como arrays, strings, Map, Set, etc.

[9] Accesibilidad:
do {
// Código a ejecutar
} while (condición);

Control de bucles



En el desarrollo de programas, es común encontrarse con situaciones donde se necesita un control más refinado sobre la ejecución de los bucles.

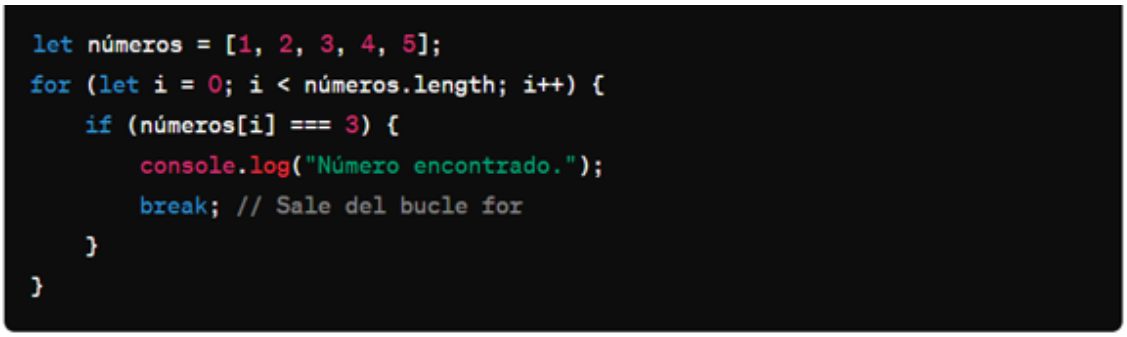
Más allá de las condiciones establecidas para while, do...while, y for, JavaScript ofrece sentencias de control adicionales como break y continue que permiten manipular el flujo de ejecución dentro de los bucles de manera más precisa. Estas herramientas son valiosas para manejar casos excepcionales, optimizar el rendimiento y mejorar la legibilidad del código.

Importancia del control de bucles

El control adecuado de los bucles es esencial para prevenir bucles infinitos, que pueden congelar o colapsar programas, y para asegurar que los bucles se comporten de manera esperada incluso en condiciones complejas o inusuales. Al utilizar break y continue, puedes hacer que tus bucles sean más flexibles y adaptativos a las necesidades de tus algoritmos y lógica de negocio.

Sentencia break

La sentencia break se utiliza para terminar la ejecución del bucle más cercano o de una sentencia switch. Al ejecutarse, break causa la salida inmediata del bucle, continuando la ejecución del código después del bucle.



```
let números = [1, 2, 3, 4, 5];
for (let i = 0; i < números.length; i++) {
    if (números[i] === 3) {
        console.log("Número encontrado.");
        break; // Sale del bucle for
    }
}
```

10

En el este ejemplo, se utiliza un bucle for para recorrer un array de números llamado números. Dentro del bucle, hay una sentencia condicional if que verifica si el elemento actual del array es igual a 3. Si la condición se cumple, es decir, si se encuentra el número 3 en el array, se imprime el mensaje "Número encontrado." y se ejecuta la sentencia break. Esta sentencia break provoca la salida inmediata del bucle for, deteniendo su ejecución, independientemente de si quedan elementos por recorrer en el array.

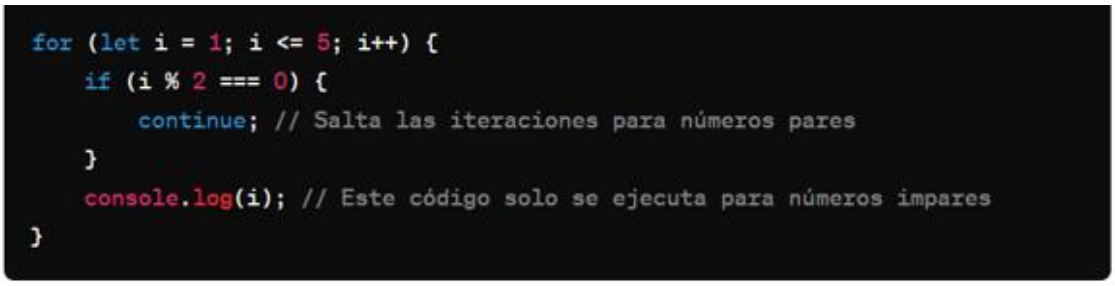
La sentencia break es especialmente útil en este contexto para evitar iteraciones innecesarias una vez que se ha encontrado el elemento deseado, mejorando así la eficiencia del código.

[10] Accesibilidad:

```
let números = [1, 2, 3, 4, 5];
for (let i=0; i < números.length; i++ ) {
    if (numeros[i] === 3) {
        console.log(numero encontrado.”);
        break; // Sale del bucle for
        }
}
```

Sentencia continue

La sentencia continue se usa para saltar el resto del código en la iteración actual del bucle y continuar con la siguiente iteración. Es útil para evitar la ejecución de ciertas partes del bucle bajo condiciones específicas, sin salir completamente del bucle.



```
for (let i = 1; i <= 5; i++) {
    if (i % 2 === 0) {
        continue; // Salta las iteraciones para números pares
    }
    console.log(i); // Este código solo se ejecuta para números impares
}
```

11

En el ejemplo proporcionado para la sentencia continue, se utiliza un bucle for para iterar desde 1 hasta 5. Dentro del bucle, hay una sentencia condicional if que verifica si el número actual i es par, utilizando el operador módulo % para determinar si el resto de la división de i por 2 es igual a 0. Si la condición se cumple (lo que significa que el número es par), se ejecuta la sentencia continue.

Esta sentencia hace que el bucle salte el resto del código dentro del bucle para la iteración actual y pase directamente a la siguiente iteración del bucle. Como resultado, el console.log(i) solo se ejecuta para los números impares, porque la ejecución del código dentro del bucle se salta para los números pares debido a la sentencia continue. Este uso de continue permite filtrar o excluir ciertas condiciones dentro de un bucle, en este caso, asegurando que solo se impriman los números impares.

---

[11] Accesibilidad:

```
for (let i = 1; i <= 5; i++) {
    if (i % 2 === 0) {
        continue; // Salta las iteraciones para los números pares
    }
    console.log(i) // Este código solo se ejecuta para números impares
}
```

Conclusión

Al explorar las estructuras de control en JavaScript, hemos desentrañado cómo las sentencias condicionales y los bucles no solo enriquecen la lógica de nuestros programas, sino que también introducen una dinámica interactiva esencial. Estas estructuras nos permiten tomar decisiones en el código (mediante if, else if, else, y el operador ternario) y repetir acciones de manera eficiente (a través de for, while, y do...while), adaptando así el comportamiento de nuestros programas a las condiciones y datos cambiantes.

Las estructuras de control son el corazón que bombea la lógica a través de nuestros programas JavaScript. Permiten que los programas respondan de manera diferente bajo diversas circunstancias, haciendo posible desde simples validaciones de datos hasta complejas iteraciones sobre grandes conjuntos de datos. Al emplear estas estructuras de manera efectiva, los desarrolladores pueden escribir código más limpio, más eficiente y reutilizable.

Conociendo las estructuras de control en JavaScript, has dado un paso crucial en tu viaje de desarrollo web, equipándote con las herramientas necesarias para construir programas interactivos y dinámicos. Te animamos a seguir practicando estos conceptos, explorando sus aplicaciones en diferentes escenarios y preparándote para los emocionantes temas que están por venir.

¡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/