

Unidad 2: Fundamentos de JavaScript

# **Condicionales y Operadores**

## **Fundamento de Condicionales**

**Introducción**

Los condicionales en JavaScript son estructuras fundamentales que permiten tomar decisiones en un programa basándose en si una o más condiciones se cumplen. Al igual que en la vida real, donde tomamos decisiones basadas en diferentes situaciones, los condicionales nos permiten ejecutar distintas partes de código dependiendo de ciertas condiciones lógicas.

### **Estructura Básica del if**

La estructura básica del if en JavaScript se utiliza para ejecutar un bloque de código solo si una condición específica es verdadera. Por ejemplo:

| if (condición) {  // Código a ejecutar si la condición es verdadera  } |
| --- |

Si la condición evaluada dentro del if resulta en true, el bloque de código asociado se ejecutará. De lo contrario, se omitirá.

### **Uso de else y else if**

Para manejar múltiples condiciones, podemos extender el uso del if con else y else if. El else permite ejecutar un bloque de código cuando la condición del if es falsa. Por ejemplo:

| if (condición) {  // Código si la condición es verdadera  } else {  // Código si la condición es falsa  } |
| --- |

Cuando hay múltiples posibles condiciones que queremos evaluar, podemos usar else if para definir condiciones adicionales:

| if (condición1) {  // Código si condición1 es verdadera  } else if (condición2) {  // Código si condición2 es verdadera  } else {  // Código si ninguna de las condiciones anteriores es verdadera  } |
| --- |

**Ejemplos Prácticos**

Consideremos un ejemplo simple para determinar si un estudiante aprueba o no un examen:

| let calificacion = 75;  if (calificacion >= 90) {  console.log("Excelente");  } else if (calificacion >= 70) {  console.log("Bueno");  } else {  console.log("Necesitas estudiar más");  } |
| --- |

Este código evalúa la calificación de un estudiante y proporciona una respuesta basada en su puntaje.

**Conclusión**

Los condicionales son herramientas poderosas en JavaScript que ayudan a dirigir el flujo de un programa. Aprendiendo a utilizar adecuadamente if, else, y else if, se puede controlar cómo se ejecuta el código en diferentes circunstancias, haciendo posible la creación de programas más dinámicos y reactivos a las condiciones de entrada.

## **Operadores en Detalle**

### **Operadores Lógicos y de Comparación en JavaScript**

**Introducción**

En JavaScript, los operadores lógicos y de comparación son herramientas esenciales que permiten evaluar condiciones y comparar valores. Estos operadores son fundamentales para controlar el flujo de ejecución en programas a través de estructuras condicionales como if, else, y bucles como while y for.

**Operadores de Comparación**

Los operadores de comparación evalúan dos valores y retornan un valor booleano (true o false) dependiendo de si la comparación es verdadera.

* == (Igual): Compara si dos valores son iguales, ignorando el tipo de dato.

| console.log(2 == '2'); // true |
| --- |

* === (Estrictamente igual): Compara si dos valores son iguales y del mismo tipo.

| console.log(2 === '2'); // false |
| --- |

* != (No igual): Compara si dos valores no son iguales, ignorando el tipo de dato.

| console.log(2 != '3'); // true |
| --- |

* !== (Estrictamente no igual): Compara si dos valores no son iguales o no son del mismo tipo.

| console.log(2 !== '2'); // true |
| --- |

* <, <=, >, >= (Menor que, Menor o igual que, Mayor que, Mayor o igual que): Utilizados para comparaciones numéricas.

| console.log(3 < 4); // true  console.log(5 >= 5); // true |
| --- |

**Operadores Lógicos**

Los operadores lógicos se usan para combinar múltiples condiciones booleanas.

&& (AND): Retorna true si ambos operandos son verdaderos.  
  
 console.log(true && false); // false

* || (OR): Retorna true si al menos uno de los operandos es verdadero.

| console.log(true || false); // true |
| --- |

* ! (NOT): Invierte el valor booleano del operando.

| console.log(!true); // false |
| --- |

**Ejemplos Prácticos**

Veamos cómo estos operadores pueden ser usados en situaciones comunes:

| let edad = 20;  let acceso = (edad >= 18) && (edad <= 30);  console.log(acceso); // true si la edad está entre 18 y 30 |
| --- |

En el ejemplo anterior, el operador && se utiliza para asegurar que ambas condiciones sobre la variable edad sean verdaderas para permitir el acceso.

| let temperatura = 35;  let advertencia = (temperatura < 0) || (temperatura > 32);  console.log(advertencia); // true si la temperatura es menor que 0 o mayor que 32 |
| --- |

Aquí, el operador || permite generar una advertencia si alguna de las condiciones climáticas es extrema.

**Conclusión**

Entender los operadores lógicos y de comparación en JavaScript es crucial para manipular el flujo de decisiones en un programa, permitiendo a los desarrolladores escribir código más flexible y robusto. Estos operadores, combinados con estructuras de control, ofrecen una poderosa herramienta para manejar la lógica de las aplicaciones.

## **Ciclos y Control de Flujos**

### **Introducción a Ciclos**

**Ciclos en JavaScript**

En JavaScript, los ciclos o bucles permiten repetir una o más instrucciones de manera eficiente hasta que se cumpla una condición específica. A continuación, exploraremos los tres tipos principales de ciclos: for, while, y do...while.

**Ciclo For**

El ciclo for es uno de los más utilizados para ejecutar un bloque de código un número determinado de veces. Su estructura incluye tres partes principales:

* **Inicialización**: Establece el estado del contador, por ejemplo, let i = 0.
* **Condición**: El ciclo se ejecuta mientras la condición sea verdadera. Por ejemplo, i < 10.
* **Actualización**: Actualiza el valor del contador después de cada iteración. Por ejemplo, i++.

**Ejemplo de Ciclo For**

| for (let i = 0; i < 5; i++) {  console.log('Número: ' + i);  } |
| --- |

Este código imprimirá los números del 0 al 4 en la consola.

**Ciclo While**

El ciclo while sigue ejecutándose mientras la condición especificada sea verdadera. A diferencia del ciclo for, la inicialización y actualización del contador suelen hacerse fuera y dentro del ciclo, respectivamente.

**Ejemplo de Ciclo While**

| let i = 0;  while (i < 5) {  console.log('Número: ' + i);  i++;  } |
| --- |

Este código también imprimirá los números del 0 al 4.

**Ciclo Do...While**

A diferencia del ciclo while, el ciclo do...while garantiza que el cuerpo del bucle se ejecute al menos una vez, ya que la condición se evalúa después de la ejecución del código dentro del bucle.

**Ejemplo de Ciclo Do...While**

| let i = 0;  do {  console.log('Número: ' + i);  i++;  } while (i < 5); |
| --- |

Este código imprimirá los números del 0 al 4, asegurando que el bloque de código se ejecute al menos una vez.

**Conclusión**

Los ciclos son herramientas poderosas en JavaScript que facilitan la escritura de código eficiente y conciso para tareas repetitivas. Cada tipo de ciclo tiene sus propias características y casos de uso óptimos, permitiendo a los desarrolladores elegir el más adecuado según la necesidad específica.

## **Controlando Ciclos**

### **Uso de break y continue**

**Introducción**

En JavaScript, los bucles son estructuras que repiten bloques de código. Dentro de estos ciclos, las declaraciones break y continue permiten manejar el flujo de ejecución de formas más flexibles y potentes. Aquí veremos cómo y cuándo utilizar estas declaraciones para controlar los ciclos eficazmente.

**Uso de break**

La declaración break se utiliza para salir de un ciclo de forma inmediata, sin importar si la condición del ciclo sigue siendo verdadera.

**Ejemplo con break**

| for (let i = 0; i < 10; i++) {  if (i === 5) {  break; // Sale del ciclo cuando i es igual a 5  }  console.log(i); // Imprime los números de 0 a 4  } |
| --- |

**Contexto de uso:** break es útil cuando necesitas detener el ciclo en una condición particular, por ejemplo, cuando se encuentra un valor específico en un array.

**Uso de continue**

La declaración continue omite la iteración actual del ciclo y continua con la siguiente, siempre y cuando la condición del ciclo siga siendo verdadera.

**Ejemplo con continue**

| for (let i = 0; i < 10; i++) {  if (i % 2 === 0) {  continue; // Omite los números pares  }  console.log(i); // Imprime los números impares entre 0 y 9  } |
| --- |

**Contexto de uso:** continue es efectivo en casos donde solo algunas condiciones específicas requieren que el ciclo se omita, como saltarse valores especiales o erróneos dentro de una secuencia de datos.

**Consideraciones Importantes**

* **Lugar de uso:** Ambas declaraciones se pueden usar en todos los tipos de ciclos (for, while, do...while).
* **Control del flujo:** Ayudan a hacer el código más claro y fácil de entender al manejar casos especiales sin anidar demasiadas condiciones.
* **Prevención de bucles infinitos:** Es crucial actualizar las variables de control de los bucles cuando se utilizan break o continue para evitar ciclos sin fin.

**Conclusión**

break y continue son herramientas poderosas que ofrecen control preciso sobre la ejecución de los ciclos en JavaScript. Su uso adecuado permite construir aplicaciones más eficientes y códigos más legibles, adaptando el flujo de los programas a cualquier necesidad específica que surja durante el desarrollo.

## **NULLISH PENDIENTE PORQUE ESTÁ EN CONSTRUCCIÓN LA SECCIÓN**