Diplomatura en

Programación FullStack Developer

GLOBALACADEMY

JavaScript
Estructuras de Control

Control de flujo

El control de flujo en programación se refiere a la decisión de cómo se ejecutan las instrucciones en un programa. Estas decisiones se toman basadas en los valores de las variables y otros estados del programa. Los principales mecanismos de control de flujo en la mayoría de los lenguajes de programación son: estructuras condicionales, bucles y saltos.

Conjunciones

En JavaScript, las conjunciones son operadores lógicos que se utilizan para combinar dos o más expresiones booleanas. Las principales conjunciones en JavaScript son:

Operador AND (&&):
•Devuelve true si ambas expresiones son verdaderas (truthy).

•Devuelve false si al menos una de las expresiones es falsa (falsy).

•Realiza una "cortocircuito", lo que significa que si la primera expresión es falsa, la segunda expresión no se evalúa.

```
const a = true;
const b = false;

console.log(a && b); // Salida: false
```

Operador OR (||):
•Devuelve true si al menos una de las expresiones es verdadera (truthy).

•Devuelve false si ambas expresiones son falsas (falsy).

•Realiza una "cortocircuito", lo que significa que si la primera expresión es verdadera, la segunda expresión no se evalúa.

```
const c = true;
const d = false;

console.log(c || d); // Salida: true
```

Operador NOT (!):

•Niega la expresión booleana, convirtiéndola en su valor opuesto.

•Si la expresión es verdadera (truthy), el operador ! la convierte en false.

•Si la expresión es falsa (falsy), el operador ! la convierte en true.

```
const e = true;
const f = !e; // f es false

console.log(f); // Salida: false
```

Estructuras Condicionales:

Estas estructuras permiten que ciertos bloques de código se ejecuten solo si se cumple una condición determinada.

if:

Evalúa una condición y, si es verdadera, ejecuta un bloque de código.

```
if (condición) {
    // Bloque de código a ejecutar si la condición es verdadera
}
```

if-else:

Si la condición es verdadera, ejecuta un bloque de código; de lo contrario, ejecuta otro bloque.

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

else if:

La estructura else if se usa en conjunto con el if para probar múltiples condiciones en secuencia. Si la condición en el if no se cumple, se evalúa la condición en el else if. Puedes tener tantos bloques else if como desees. Al final, puedes optar por agregar un bloque else que se ejecutará si ninguna de las condiciones anteriores se cumplió. Estructura:

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

Ejemplo:

Supongamos que queremos categorizar a las personas según su edad:

```
let edad = 25;

if (edad < 13) {
    console.log("Edad de niño");
} else if (edad >= 13 && edad < 20) {
    console.log("Sos un adolescente");
} else if (edad >= 20 && edad < 30) {
    console.log("Sos un joven adulto");
} else {
    console.log("Sos un adulto");
}
```

switch :

El switch es una estructura de control que permite realizar una selección múltiple de bloques de código basándose en una expresión. Es especialmente útil cuando tienes múltiples condiciones o casos para evaluar y quieres evitar el uso repetido de condicionales if...else if.

Estructura Básica:

El switch toma una expresión y la compara con múltiples valores posibles (llamados casos). Cada caso tiene un bloque de código asociado que se ejecuta si la expresión coincide con el valor del caso. Aquí tienes la sintaxis básica:

```
switch (expresión) {
    case valor1:
        // Código para ejecutar si la expresión coincide con valor1
        break;
    case valor2:
        // Código para ejecutar si la expresión coincide con valor2
        break;
    case valor3:
        // Código para ejecutar si la expresión coincide con valor3
        break;
    default:
    // Código para ejecutar si la expresión no coincide con ningún caso anterior
}
```

Explicación de la Sintaxis:

switch (expresión):
La expresión (por ejemplo, una variable o función que devuelve un valor) se evalúa inicialmente. Luego, el switch compara este valor con los valores definidos en los distintos casos.

case valor:
Define un caso específico que se compara con la expresión inicial. Si coincide, el código asociado a este caso se ejecuta.

break:
Se utiliza al final de cada caso para salir de la estructura switch. Sin el break, el flujo de ejecución continuaría al siguiente caso, ejecutando su código incluso si la expresión no coincide con él.

default:
Proporciona un bloque de código alternativo que se ejecuta si ninguno de los casos anteriores coincide con la expresión. El default es opcional, pero es una buena práctica incluirlo para manejar todas las posibilidades.

Ejemplo Práctico:

Imaginemos que queres implementar una función que devuelva el nombre del día de la semana en función de un número del 1 al 5:

```
let dia = 1

switch (dia) {
    case 1:
        dia = "Lunes";
        break
    case 2:
        dia = "Martes";
        break
    case 3:
        dia = "Miércoles";
        break
    case 4:
        dia = "Jueves";
        break
    case 5:
        dia = "Viernes";
        break
    default:
        dia = "Número no válido";
        break
}
console.log(dia);
```

Bucles:

Los bucles permiten la repetición de un bloque de código mientras se cumpla una condición.

for:

Ejecuta un bloque de código un número determinado de veces:

```
for (inicialización; condición; actualización) {
    // Bloque de código a repetir
}
```

Inicialización: Aquí estableces un punto de inicio para tu bucle. Por lo general, se utiliza para inicializar un contador. Por ejemplo: let i = 0;.

Condición: Esta es una expresión que se evalúa antes de cada iteración del bucle. Si resulta ser true, el bucle continuará ejecutándose. Si es false, el bucle se detendrá. Por ejemplo: i < 10;.

actualización: Esta expresión se ejecuta al final de cada iteración del bucle. Comúnmente se utiliza para actualizar el valor del contador. Por ejemplo: i++ (que es un atajo para i = i + 1).

Ejemplo clásico de un bucle for:

Los bucles permiten la repetición de un bloque de código mientras se cumpla una condición.

```
for (let i = 0; i < 5; i++) {
    console.log("El valor de i es: " + i);
}
```

Lo que sucede paso a paso es lo siguiente:

1. Antes de la primera iteración, i se inicializa a 0.
2. Se verifica si i es menor que 5. Dado que 0 es menor que 5, la condición es verdadera.
3. Se ejecuta el bloque de código dentro del bucle, que imprime "El valor de i es: 0".
4. Al final de la iteración, i se incrementa en 1 debido a la expresión i++.
5. El bucle vuelve a la condición. Ahora, i es 1, que todavía es menor que 5, por lo que el bucle se repite.
6. Este proceso continúa hasta que i es 5. En ese momento, la condición i < 5 ya no se cumple, por lo que el bucle se detiene.

El bucle for es extremadamente versátil. Si bien el ejemplo anterior muestra un incremento simple, puedes modificar la inicialización, la condición y el incremento/decremento para adaptarse a muchas situaciones diferentes según las necesidades

Bucle for...of

El bucle for...of fue introducido en ES6 (ECMAScript 2015) para iterar sobre elementos iterables como arrays, strings, maps, sets, y más, de una manera más sencilla y directa. La sintaxis es:

```
for (variable of iterable) {
    // Bloque de código a ejecutar
  }
```

- Variable: En cada iteración, a esta variable se le asigna el valor del elemento actual del iterable.
- Iterable: El objeto cuyos elementos se van a iterar (por ejemplo, un array o un string). for...of es particularmente útil para iterar sobre colecciones de datos donde te interesa el valor de cada elemento y no su índice o clave.

Bucle for...in

El bucle for...in itera sobre todas las propiedades enumerables de un objeto, proporcionando una manera fácil de recorrer todas sus propiedades. La sintaxis es:

```
for (variable in objeto) {
    // Bloque de código a ejecutar
}
```

- Variable: En cada iteración, a esta variable se le asigna el nombre de la propiedad actual que se está procesando.

- Objeto: El objeto cuyas propiedades se van a iterar. Es importante tener en cuenta que for...in no garantiza un orden específico al recorrer las propiedades del objeto y puede no ser la elección adecuada para iterar sobre arrays donde el orden es importante.

Ejemplos

```
for (let i = 1; i <= 5; i++) {
    console.log(i);
}
const persona = {
    nombre: "Juan",
    edad: 30,
    ciudad: "Buenos Aires"
};
for (let clave in persona) {
    console.log(`${clave}: ${persona[clave]}`);
}
const frutas = ["manzana", "banana", "cereza"];
for (let fruta of frutas) {
    console.log(fruta);
}
```

while:

El bucle while es otra estructura fundamental de control de flujo en programación. A diferencia del bucle for, que generalmente se utiliza cuando sabes cuántas veces quieres que se ejecute un bloque de código, el bucle while es más apropiado cuando quieres que el bloque de código se ejecute mientras se cumpla una condición, pero no sabes cuántas veces exactamente se repetirá.

La sintaxis del bucle while es la siguiente:

```
while (condición) {
    // bloque de código a repetir
}
```

Desglosemos el componente principal:

Condición: Esta es una expresión que se evalúa antes de cada iteración del bucle. Si la condición es true, el bucle continuará ejecutándose. Si es false, el bucle se detendrá.

Ejemplo:

```
let contador = 0;
while (contador < 5) {
    console.log("El valor del contador es: " + contador);
    contador++;
}
```

1. Antes de la primera iteración, el contador se inicializa a 0.

2. Se verifica si contador es menor que 5. Dado que 0 es menor que 5, la condición es verdadera.

3. Se ejecuta el bloque de código dentro del bucle, que imprime "El valor del contador es: 0".

4. Al final de la iteración, el contador se incrementa en 1.

5. El bucle regresa a la condición. Ahora, contador es 1, que todavía es menor que 5, por lo que el bucle se repite.

6. Este proceso continúa hasta que el contador es 5. En ese punto, la condición contador < 5 ya no se cumple y el bucle se detiene.

Una advertencia importante con el bucle while es asegurarte de que la condición del bucle eventualmente se vuelva false. De lo contrario, entrarás en un bucle infinito, lo que puede causar que tu programa o aplicación se cuelgue.

En resumen, el bucle while es útil cuando quieres repetir un bloque de código basado en una condición, pero no sabes de antemano cuántas veces se necesitará ejecutar ese bloque. Solo debes tener cuidado de evitar bucles infinitos al usarlo.

Bucle do while

El bucle do...while es similar al bucle while, con la diferencia principal de que garantiza que el bloque de código se ejecute al menos una vez. Esto se debe a que la condición se evalúa después de la ejecución del bloque de código.

```
do {
    // Bloque de código a ejecutar
} while (condición);
```

Primero se ejecuta el bloque de código dentro del do, y luego se evalúa la condición en el while. Si la condición es verdadera, el bloque de código se ejecuta nuevamente. Este proceso se repite hasta que la condición se evalúa como falsa.

Ejemplo:

```
let entrada;

do {
    entrada = prompt("Introduce una palabra (escribe 'stop' para detener):");
          // Solicita la entrada del usuario
    console.log("Has introducido:", entrada); // Muestra lo que se introdujo
} while (entrada.toLowerCase() !== "stop"); // Comprueba si la entrada es "stop"
```