

Teoría JavaScript

Las estructuras repetitivas y de control son fundamentales en la programación, ya que nos permiten controlar el flujo de ejecución de un programa y repetir ciertas acciones. Estas estructuras son especialmente útiles cuando necesitamos realizar tareas de manera repetitiva o tomar decisiones basadas en ciertas condiciones. Por ejemplo, supongamos que tenemos una lista de nombres y queremos mostrarlos uno por uno. En lugar de escribir instrucciones individuales para cada nombre, podemos utilizar una estructura repetitiva como el bucle `for` o `while` para recorrer la lista y mostrar cada nombre automáticamente. Esto nos ahorra tiempo y esfuerzo, ya que no tenemos que repetir manualmente las mismas instrucciones una y otra vez.

Las estructuras repetitivas y de control también nos permiten tomar decisiones en función de ciertas condiciones. Por ejemplo, imaginemos que estamos construyendo un programa de chat y queremos que se muestre un mensaje de bienvenida solo si el usuario está autenticado. Podemos utilizar una estructura de control como una declaración `if` para evaluar si el usuario está autenticado y, en caso afirmativo, mostrar el mensaje de bienvenida. De esta manera, podemos adaptar el comportamiento del programa en función de la situación actual. Las estructuras de control nos brindan la flexibilidad necesaria para tomar decisiones y responder dinámicamente a diferentes escenarios.

Operadores de comparación

Los operadores de comparación en JavaScript se utilizan para comparar valores.

Igualdad (`==`): Comprueba si dos valores son iguales en valor.

```
console.log(3 == 3); // true
console.log('3' == 3); // true
```

Desigualdad (`!=`): Comprueba si dos valores no son iguales en valor.

```
console.log(3 != '3'); // true
console.log(4 != 4); // true
```

Igualdad estricta (===): Comprueba si dos valores son iguales en tipo y valor.

```
console.log(3 === 3); // true
console.log('3' === 3); // false
```

Desigualdad estricta (!==): Comprueba si dos valores no son iguales en tipo o valor.

```
console.log(3 !== '3'); // true
console.log(4 !== 4); // false
```

Mayor que (>): Comprueba si un valor es mayor que otro.

```
console.log(5 > 3); // true
```

Menor que (<): Comprueba si un valor es menor que otro.

```
console.log(2 < 5); // true
```

Mayor o igual que (>=): Comprueba si un valor es mayor o igual que otro.

```
console.log(5 >= 5); // true
console.log(5 >= 4); // true
```

Menor o igual que (<=): Comprueba si un valor es menor o igual que otro.

```
console.log(3 <= 3); // true
console.log(2 <= 3); // true
```

Operadores lógicos

Los operadores lógicos en JavaScript se utilizan para evaluar condiciones.

AND lógico (&&): Devuelve true si ambos operandos son verdaderos.

```
console.log(true && true); // true
console.log(true && false); // false
```

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

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

NOT lógico (!): Invierte el valor de verdad del operando.

```
console.log(!true); // false
console.log(!false); // true
```

Condicionales en Javascript

Al igual que en otros lenguajes, como en Java, , existen los condicionales que nos van a ayudar a modificar el flujo de ejecución del programa.

IF

El condicional if es un condicional lógico que evalúa el camino a tomar en base a la evaluación de una condición. Supongamos el siguiente ejemplo, mi sobrino quiere subirse a una montaña rusa, pero para ello tiene que aprobar las dos siguientes condiciones: tener más de 18 años y medir más de 160 cm. La evaluación de esas dos condiciones da por verdadero se podrá subir de lo contrario no podrá.

```
let edad = 15;

let altura = 166;

if(edad>18 && altura>160){

  console.log("Puedes subirte :D");

}else{

  console.log("No te podes subir");

}
```

Como se puede ver, si la condición a evaluar se cumple, es decir, da verdadero, mostrará el mensaje *"Puedes subirte :D"*, en caso de que de falso mostrará *"No te puedes subir "*. **Por otra parte, JavaScript permite también agregar la condición else if**

```
if(a == 2){

  console.log("a es igual a 2");

}else if(a < 2){
```

```
    console.log("a es menor que 2");  
  }else{  
    console.log("a es mayor que 2");  
  }
```

IF Ternario

El if ternario nos permite resolver en una línea una expresión lógica asignando un valor. Proviene del lenguaje C, donde se escriben muy pocas líneas de código y donde cuanto menos escribamos más elegantes seremos. Lo veremos rápidamente, pues la única razón que lo veamos es para que sepan que existe y si lo encuentran en alguna ocasión sepan identificarlo y cómo funciona.

Variable = (condición) ? valor1 : valor2

Este ejemplo no solo realiza una comparación de valores, además asigna un valor a una variable. Lo que hace es evaluar la condición (colocada entre paréntesis) y si es positiva asigna el valor1 a la variable y en caso contrario le asigna el valor2.

Veamos un ejemplo:

```
let momento = (hora_actual < 12) ? "Antes del mediodía" : "Después del  
mediodía"
```

SWITCH

La declaración switch evalúa una expresión, **comparando el valor de esa expresión con una instancia case**, y ejecuta declaraciones asociadas a ese case, así como las declaraciones en los case que siguen.

El programa primero busca la primera instancia case cuya expresión se evalúa con el mismo valor de la expresión de entrada (usando comparación estricta, ===) y luego transfiere el control a esa cláusula, ejecutando las declaraciones asociadas. Si no se encuentra una cláusula de case coincidente, el programa busca la cláusula default opcional, y si se encuentra, transfiere el control a esa instancia, ejecutando las declaraciones asociadas.

La declaración `break` es opcional y está asociada con cada etiqueta de `case` y asegura que el programa salga del `switch` una vez que se ejecute la instrucción coincidente y continúe la ejecución en la instrucción siguiente. Si se omite el `break`, el programa continúa la ejecución en la siguiente instrucción en la declaración de `switch`.

```
switch (expr) {  
  case 'Naranjas':  
    console.log('El kilogramo de naranjas cuesta $0.59.');
```

`break;`

```
  case 'Manzanas':  
    console.log('El kilogramo de manzanas cuesta $0.32.');
```


`break;`

```
  case 'Papayas':  
    console.log('El kilogramo de papayas cuesta $2.79.');
```

`break;`

```
  default:  
    console.log('Lo siento, por el momento no tenemos de ' + expr + '');
```

`}`

 **TIP:** Es altamente recomendable consultar la documentación oficial para profundizar en los conceptos. Además, encontrarás una variedad de ejemplos que facilitarán la comprensión práctica de lo aprendido. Te proporcionamos el [enlace directo](#).

Estructuras Repetitivas

Veamos cómo son las estructuras repetitivas en JavaScript.

WHILE

Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia

```
let a = 0;
```

```
while(a != 10){  
    console.log(++a);  
}
```

DO WHILE

La sentencia (hacer mientras) crea un bucle que ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa. La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez.

```
let a = 0;  
  
do{  
    console.log(++a);  
}while(a!=10);
```

FOR

El bucle FOR se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute. La sintaxis del bucle for va a ser la misma que en Java:

```
for ([expresion-inicial]; [condicion]; [expresion-final]){ }  
  
for(let i = 0; i < 10; i++){  
    console.log("El valor de i es " + i);  
}
```

💡 En el ámbito de las estructuras repetitivas en programación, existen etiquetas y palabras clave como 'break', 'label' y 'continue', aunque en la actualidad su uso no es tan común. Es importante señalar que su aplicación indebida o excesiva puede disminuir la legibilidad del código y aumentar la probabilidad de errores. Si te interesa, te animamos a investigar más sobre estas construcciones para comprender su funcionamiento y considerar cuidadosamente su implementación en el código.

