Universidad Nacional de Rosario

Argentina programa 4.0

Programación
Web Full Stack PHP

Loops

La importancia de los bucles

Los bucles

En la programación, la repetición es un proceso fundamental que nos permite ejecutar una serie de instrucciones múltiples veces. Para automatizar tareas repetitivas de manera eficiente, JavaScript ofrece diferentes tipos de bucles que evitan tener que repetir el mismo código manualmente una y otra vez. Esto ahorra tiempo, reduce errores y hace que el código sea más conciso y fácil de mantener.

Los bucles son estructuras de control que permiten repetir un bloque de código hasta que se cumpla una condición determinada. En JavaScript, hay diferentes tipos de bucles disponibles. Cada tipo de bucle tiene su propia sintaxis y se utiliza en situaciones específicas.

- for
- do...while
- while
- for...in
- for...of

Pueden ser utilizados para recorrer elementos de un array, procesar listas de datos, realizar cálculos iterativos, etc. Al utilizar bucles de manera efectiva, puedes reducir la redundancia en tu código y mejorar la eficiencia de tus programas.

El bucle for

Es una estructura de control utilizada para repetir un bloque de código un número específico de veces. Es especialmente útil cuando se conoce de antemano la cantidad exacta de repeticiones que se desea realizar.

Sintaxis:

- **Inicialización**: Se define una variable contadora a la cual, por buenas prácticas, se la denomina "**i**" y se le asigna un valor inicial.
- **Condición**: Se especifica una condición que debe ser verdadera para que el bucle se ejecute en cada iteración. Mientras esta condición sea verdadera, el bucle continuará ejecutándose.
- **Expresión de actualización**: Se utiliza para modificar la variable luego de cada iteración. Por lo general, se incrementa o decrementa el valor de la variable para asegurar su finalización.
- **Bloque de código** a ejecutar en cada iteración

```
for(inicializacion; condicion; actualizacion) {
    // Bloque de código a ejecutar por iteración
}
```

A continuación, se muestra un ejemplo de bucle "for" que imprime los números del 0 al 9 en la consola:

```
for (let i = 0; i < 10; i++) {
    console.log(i);
}
```

En el ejemplo anterior, `let i = 0` inicializa la variable `i` con el valor 0. La condición `i < 10` establece que el bucle se ejecutará mientras `i` sea menor a 10. Después de cada iteración, `i++` incrementa el valor de `i` en 1.

El bucle imprimirá los números del 0 al 9 en la consola, ya que en cada iteración el valor de `i` se mostrará mediante `console.log(i)`.

El bucle while

Es otra estructura de control utilizada para repetir un bloque de código mientras se cumpla una condición determinada. A diferencia del bucle "for", el bucle "while" se ejecuta mientras la condición sea verdadera y no tiene una contabilización explícita de iteraciones.

Sintaxis:

- **Condición**: Se especifica una condición que se evalúa antes de cada iteración. Si la condición es verdadera, el bloque de código dentro del bucle se ejecuta. Si la condición es falsa, el bucle se detiene y la ejecución continúa con el código después del bucle.
- **Bloque de código** a ejecutar en cada iteración.

```
while(condicion){
    // Bloque de código a ejecutar por iteración
}
```

A continuación, se muestra un ejemplo de bucle "while" que imprime los números del 0 al 9 en la consola:

```
let contador = 0;

while (contador < 10) {
    console.log(contador);
    contador++;
}
```

En el ejemplo anterior, se inicializa la variable **contador** con con el valor 0. La condición **i < 10** se evalúa antes de cada iteración. Si la condición es verdadera, se ejecuta el bloque de código dentro del

bucle, que incluye **console.log(contador)** para mostrar el valor actual de **contador**. Luego, **contador++** se utiliza para incrementar el valor de **contador** en 1.
El bucle imprimirá los números del 0 al 9 en la consola, ya que mientras **contador** sea menor 10, el bloque de código dentro del bucle se seguirá ejecutando.

Es importante tener en cuenta que el bucle "while" puede **no ejecutarse nunca** si la condición especificada **nunca es verdadera**. Esto significa que, si la condición es falsa desde el principio, el bloque de código dentro del bucle no se ejecutará en absoluto.

Cuidado con bucles infinitos: Si la condición del bucle nunca se evalúa como falsa, el bucle puede continuar ejecutándose infinitamente, lo que se conoce como un "bucle infinito". Los bucles infinitos pueden causar que un programa se bloquee o entre en un estado de ejecución indefinida, por lo que debes tener cuidado al diseñar la condición del bucle y asegurarte de que eventualmente se vuelva falsa.

El bucle do while

Es una estructura de control utilizada para repetir un bloque de código mientras se cumpla una condición determinada. A diferencia del bucle "while", el bucle "do-while" garantiza que el bloque de código se ejecute al menos una vez, incluso si la condición es inicialmente falsa.

Sintaxis:

- **Bloque de código**: Se especifica un bloque de código que se ejecutará al menos una vez y luego se repetirá mientras se cumpla la condición.
- **Condición**: Después de la ejecución del bloque de código, se evalúa una condición. Si la condición es verdadera, el bucle se repetirá y el bloque de código se ejecutará nuevamente. Si la condición es falsa, el bucle se detendrá y la ejecución continuará con el código después del bucle.
- **Bloque de código** a ejecutar en cada iteración.

```
do {
    // Bloque de código a ejecutar por iteración
} while (condicion);
```

A continuación, se muestra un ejemplo que imprime los números del 0 al 9 en la consola:

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

En el ejemplo anterior, se inicializa la variable **contador** con el valor 0. El bloque de código dentro del bucle se ejecuta primero, lo que incluye **console.log(contador)** para mostrar el valor actual de **contador**. Luego, **contador++** se utiliza para incrementar el valor de **contador** en 1.
Después de la ejecución del bloque, la condición **i < 10** se evalúa. Si la condición es verdadera, el bucle se repetirá y el bloque se ejecutará nuevamente. Si la condición es falsa, el bucle se detendrá y la ejecución continuará con el código después del bucle.
Como resultado imprimirá los números del 0 al 9 en la consola y garantiza que el bloque de código se ejecute al menos una vez.

El bucle for of

Es una estructura de control introducida en ECMAScript 6 (ES6) que se utiliza para iterar sobre elementos de una estructura iterable, como un array, una cadena de texto, un objeto iterable, etc. Proporciona una forma sencilla y concisa de recorrer todos los elementos de una colección. En la primera iteración del bucle, la variable especificada (por ejemplo, **elemento**) toma el valor del primer elemento del iterable. El bucle se ejecuta una vez por cada elemento del iterable. Después de cada

iteración, la variable toma el valor del siguiente elemento y el bloque de código dentro del bucle se ejecuta. El bucle **for...of** se detiene automáticamente después de que se hayan recorrido todos los elementos del iterable. No es necesario especificar una condición de finalización

Sintaxis:

- **variable**: Es el nombre de la variable que se utilizará para representar cada elemento del iterable en cada iteración. Debes declarar esta variable utilizando **let o const** antes de iniciar el bucle.
- **of**: La palabra clave **of** se utiliza para indicar que se recorrerán los elementos del iterable.
- **iterable**: El iterable es la estructura sobre la cual se realizará la iteración. Puede ser cualquier estructura iterable, como un arreglo, una cadena de texto, etc.
- **Bloque de código**: Es el bloque de código que se ejecutará en cada iteración del bucle. Puedes incluir cualquier instrucción o conjunto de instrucciones dentro de las llaves **{}**. Este bloque se ejecutará una vez por cada elemento del iterable.

```
for(let elemento of miLista) {
    // Bloque de código a ejecutar por iteración
}
```

A continuación vemos un ejemplo que imprime los números del 0 al 9 en la consola:

```
let miLista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

for (let elemento of miLista) {
    console.log(elemento);
}
```

El bucle **for...of** recorre cada elemento del array miLista. En cada iteración, la variable **elemento** contiene el valor del elemento actual, que se muestra en la consola utilizando **console.log(elemento)**.

For In

El bucle **for...in** se utiliza para iterar sobre las propiedades enumerables de un objeto, por cada preopiedad ejecuta el bloque de código. A diferencia del bucle **for...of** que se utiliza para iterar sobre elementos de una estructura iterable, el bucle **for...in** se enfoca en recorrer las claves (o keys) de un objeto.

Sintaxis:

- **clave**: Es una variable que se utiliza para almacenar el nombre de cada propiedad del objeto a medida que se recorre el bucle. Puedes elegir cualquier nombre válido para esta variable. Por lo general, se utiliza el término "clave" o "propiedad".
- **objeto**: Es el objeto sobre el cual se va a iterar y se van a obtener las propiedades. Puede ser cualquier objeto válido en JavaScript, incluyendo objetos predefinidos (como **Array**, **Object**, etc.) o objetos definidos por el usuario.

```
for (let clave in objeto) {
    // Bloque de código a ejecutar por iteración
}
```

En cada iteración del bucle, la variable **clave** toma el nombre de una propiedad del objeto. Dentro del bloque de código, puedes acceder al valor correspondiente utilizando la notación de corchetes **objeto[clave]** o la notación de punto **objeto.clave**.

Ejemplo

```
const persona = {
    nombre: "Juan",
    edad: 30,
    profesion: "Programador",
};

for (let clave in persona) {
    console.log(clave + ": " + persona[clave]);
}
```

En este caso, el bucle **for...in** itera sobre las propiedades del objeto **persona**. En cada iteración, la variable **clave** contiene el nombre de la propiedad, y mediante **persona[clave]** se accede al valor correspondiente. En el ejemplo, se mostrará en la consola cada clave seguida de su valor.

Controlar el flujo de ejecución en bucles con break y continue

Break

Se utiliza la instrucción break para terminar un bucle. Termina inmediatamente el bucle while, do-while, for que lo rodea y transfiere el control a la siguiente declaración.

Cuando se encuentra una instrucción "break" dentro de un bucle, el programa sale del bucle por completo y continúa con la siguiente línea de código después del bucle.

El código anterior al llegar al índice 5, termina la ejecución del for, mostrando en consola 0, 1, 2, 3, 4 hasta 5.

```
for (let i = 0; i < 10; i++) {
    console.log(i);
    if (i === 5) {
        break;
    }
}
```

El break también es utilizado en estructuras condicionales como el switch, ver material Lógica.

Continue

La declaración continue se puede utilizar para reiniciar una declaración while, do-while, for.
Cuando se utiliza *continue*, termina la iteración actual del ciclo while, do-while, o for más interno y continúa la ejecución del ciclo con la siguiente iteración. En contraste con la declaración break, continue no finaliza la ejecución del ciclo por completo. En un ciclo while, salta de vuelta a la condición. En un ciclo for, salta a la expresión incremental.
Ejemplo:

```
for (let i = 0; i < 10; i++) {
    if (i % 2 === 0) {
        continue;
    }
    console.log(i);
}
```

Dentro del bucle, se encuentra una estructura **if** que verifica si **i** es divisible por 2, lo cual se evalúa utilizando el operador **%** para calcular el resto de la división. Si **i** es par, es decir, si el resto de **i** dividido por 2 es igual a 0, se ejecuta la instrucción **continue**.

Cuando se encuentra **continue**, se salta la iteración actual y se pasa a la siguiente iteración sin ejecutar el código restante dentro del bloque.

En este caso, cuando **i** es par, se aplica **continue** y se omiten las instrucciones **console.log(i)**, lo que significa que no se mostrarán en la consola los números pares. El bucle continuará con la siguiente iteración.

Como resultado, en la consola se mostrarán los números impares del 1 al 9, ya que todas las iteraciones con valores pares fueron omitidas debido al uso de **continue**.

Este ejemplo ilustra cómo **continue** puede ser utilizado para controlar qué iteraciones se ejecutan en un bucle y cuáles se omiten según una determinada condición.