Logotipo

El contenido generado por IA puede ser incorrecto.

Santiago Yosa González

**Ingeniero Sistemas de Información**



Imagen que contiene foto, frente, caja, tabla

El contenido generado por IA puede ser incorrecto.

**Generalidades de Javascript**

**Unidad I**

**Ciclos en JavaScript:**

Estructuras

* For
* while
* do-while.

**Situación**

En la sesión anterior, aprendimos a usar if para tomar una decisión. Por ejemplo, podríamos comprobar si la nota de *un* estudiante es mayor a 3.0 para saber si aprobó.

Ahora, ¿qué pasa si tenemos un salón con 40 estudiantes? ¿Escribimos 40 bloques if diferentes? if (estudiante1.nota > 3.0) { ... } if (estudiante2.nota > 3.0) { ... } ... if (estudiante40.nota > 3.0) { ... }

Eso es ineficiente, imposible de mantener y viola el principio de "No Te Repitas" (DRY - Don't Repeat Yourself).

Necesitamos una forma de decirle a JavaScript:

"Toma esta lista de estudiantes y **repite** esta misma lógica de if para **cada uno** de ellos".

Esa instrucción de "repetir" es un **ciclo** o **bucle**. Los bucles son el motor de nuestras aplicaciones; son la maquinaria que nos permite procesar miles de datos con unas pocas líneas de código.

**El Concepto de Iteración**

Un ciclo (o bucle) es una estructura de control que repite un bloque de código mientras una condición sea **verdadera (true)**

**El Ciclo for (La Iteración Definida)**

El ciclo for es la estructura de repetición más común. Se usa cuando **sabemos de antemano cuántas veces** queremos que se repita una acción. (Ej. "Repite esto 10 veces", "Repite esto por cada uno de los 40 estudiantes").

Un ciclo for es como un entrenador que te dice: "Corre 5 vueltas a la pista".

1. **Punto de Partida:** Empiezas en la vuelta 0 (o 1).
2. **Condición para Seguir:** ¿Has corrido *menos de* 5 vueltas? Si es sí, sigue corriendo.
3. **Acción por Vuelta:** Al final de cada vuelta, anotas 1 vuelta más en tu contador.

**Anatomía del Código**

El for se define con tres partes clave dentro de los paréntesis, separadas por punto y coma: for (inicialización; condición; incremento) { ... }

// Este ciclo imprimirá los números del 0 al 4

for (let i = 0; i < 5; i++) {

console.log('El número es: ' + i);

}

* **1. let i = 0; (La Inicialización):** Es el "Punto de Partida". Creamos una variable (¡usando let!) que actuará como nuestro contador. Esto solo se ejecuta **una vez** al inicio.
* **2. i < 5; (La Condición):** Es la "Regla para Continuar". Antes de *cada* vuelta, JavaScript revisa esta condición. Si es true, ejecuta el bloque de código. Si es false, el ciclo se detiene.
* **3. i++ (El Incremento):** Es el "Paso". Se ejecuta **después** de cada vuelta. i++ (de la Sesión 1) suma 1 a nuestro contador, acercándonos al final.

**Profundización**

¿Por qué let y no var? Si usáramos var i, esa variable i se "fugaría" del ciclo y existiría en todo nuestro código, pudiendo causar errores. Al usar let i, la variable i **solo existe dentro** del bloque {...} del ciclo, lo cual es mucho más seguro y predecible.

**El Ciclo while**

Usamos un while cuando **no sabemos cuántas veces** necesitamos repetir. La repetición no depende de un contador, sino de que una condición externa siga siendo verdadera.

**Mientras** el jugador tenga vidas (vidas > 0), sigue jugando. No sabes si el juego durará 1 minuto o 3 horas. El ciclo continúa *mientras* la condición se cumpla.

**¡Peligro! El Bucle Infinito:**

El while es poderoso pero peligroso. Si la condición *nunca* se vuelve false, el código se repetirá para siempre, congelando el navegador. **Algo dentro del while debe, eventualmente, cambiar la condición.**

**Anatomía del Código**

let vidas = 3;

while (vidas > 0) {

console.log(`¡Tienes ${vidas} vidas!`);

// Simulamos que el jugador pierde una vida

// ESTA LÍNEA ES CRUCIAL. Modifica la condición.

vidas--; // Sin esto, sería un bucle infinito.

}

console.log('Game Over');

**El Ciclo do...while**

Es casi idéntico al while, con una sola diferencia crucial: el do...while **siempre se ejecuta al menos una vez**.

***Caso de uso:***

Un menú de opciones donde primero muestras el menú y luego preguntas si quiere pedir algo del menú.

Se usa en situaciones donde necesitas que el código se ejecute una vez antes de comprobar si debe repetirse.

**Anatomía del Código**

let contrasena;

do {

// Este bloque se ejecuta SIEMPRE al menos una vez

contrasena = prompt('Introduce tu contraseña:');

} while (contrasena !== '12345'); // La condición se revisa DESPUÉS

console.log('¡Bienvenido!');

**El Bucle for...of**

Este es el sucesor moderno del ciclo for clásico cuando quieres recorrer un **Arreglo**. Es mucho más limpio y menos propenso a errores.

**Para qué sirve:**

Para iterar sobre los **valores** de un objeto "iterable" (como un Arreglo, un String, un Mapa, etc.).

**Casos de Uso**

* **Ciclo for (clásico):** Eres un repartidor al que le dicen: "Ve a la casa #0, luego a la casa #1, luego a la #2...". Te importa mucho el **índice** (el número de la casa).
* **Ciclo for...of:** Eres un repartidor al que le dan una lista de clientes. Vas al "cliente A", luego al "cliente B". No te importa si son la casa #0 o la #1(indice), solo te importa el **valor** (el cliente).

**Anatomía del Código:**

const miArray = ['manzana', 'banana', 'naranja'];

// El 'for...of'

for (const fruta of miArray) {

// 'fruta' toma el VALOR de cada elemento en cada vuelta

console.log(fruta);

}

// Resultado:

// "manzana"

// "banana"

// "naranja"

**Cuándo usarlo:** Es la forma **preferida y moderna** de recorrer un arreglo cuando solo te importa el valor de cada elemento y no necesitas saber su posición (el índice i).

**El Bucle for...in (El Iterador de Propiedades de Objetos)**

Este bucle se ve similar, pero es **fundamentalmente diferente** y es una fuente común de confusión.

**Para qué sirve:**

Para iterar sobre las **propiedades** (las "claves" o *keys*) de un **Objeto**.

A este bucle no le importa quién vive en la casa (valor), le importa el nombre de las habitaciones (clave). Si le das un objeto persona, te dirá que tiene una habitación "nombre", una "edad" y una "profesión".

**¡Advertencia Importante!**

Nunca uses for...in para recorrer un **Arreglo**. Técnicamente "funciona", pero lo hará de forma incorrecta:

1. Iterará sobre los **índices** ('0', '1', '2') como si fueran strings, no sobre los valores.
2. Puede iterar sobre propiedades inesperadas que no son parte de los datos.

**Resumen de la Diferencia:**

* for...of -> Para **valores** de **Arreglos**.
* for...in -> Para **claves** de **Objetos**.

**Anatomía del Código:**

const persona = {

nombre: 'Ana',

edad: 30,

profesion: 'Doctora'

};

// El 'for...in'

for (const clave in persona) {

// 'clave' toma el NOMBRE de la propiedad en cada vuelta

console.log(clave); // 'nombre', 'edad', 'profesion'

// Para ver el valor, debes usar la clave

console.log(persona[clave]); // 'Ana', 30, 'Doctora'

}

**Mención Honorífica:**

**.forEach() (El Método de Array)**

No es un "bucle" en el sentido de una sentencia de JavaScript, sino un **método** que tienen los arreglos. Es la forma "funcional" de hacer un for...of.

const miArray = ['manzana', 'banana', 'naranja'];

// El .forEach() recibe una función callback

miArray.forEach(function(fruta) {

console.log(fruta);

});

**Control de Ciclos (break y continue)**

A veces necesitamos alterar el flujo normal de un ciclo.

* **break (La Salida de Emergencia):** Termina el ciclo **inmediatamente**, sin importar si la condición se sigue cumpliendo.
  + **Uso:** Estoy buscando un nombre en una lista de 1000. Lo encontré en la posición 50. ¡No necesito revisar los 950 restantes! ¡break!
* **continue (Saltar esta Vuelta):** Detiene la iteración **actual** y salta directamente a la siguiente (al i++ en un for).
  + **Uso:** Estoy procesando 100 emails. El email actual es spam. No lo proceso (continue) y salto al siguiente email.

**Preguntas**

**¿Cuál es la diferencia real entre for y while?**

Es la pregunta más importante. Usa for cuando tengas una **iteración definida** (sabes cuántas veces, ej: 10 vueltas, o una vez por cada elemento de una lista). Usa while cuando tengas una **iteración indefinida** (no sabes cuántas veces, ej: "hasta que el usuario escriba 'salir'", "mientras el jugador tenga vidas").

**Mi navegador se congeló cuando usé un while. ¿Qué pasó?**

Creaste un **bucle infinito**. Esto sucede cuando la condición de tu while empieza siendo true y nada dentro del bucle hace que se vuelva false. Por ejemplo: let i = 0; while (i < 10) { console.log(i); }. La variable i nunca cambia, siempre es 0, por lo que 0 < 10 siempre es true. La solución es añadir un incremento (i++) dentro del bucle.

**¿Por qué siempre se usa la variable i?**

Es una convención histórica que viene de "Índice" o "Iterador". Si tienes ciclos anidados (uno dentro de otro), se suele usar j, luego k. Sin embargo, puedes llamarla como quieras (contador, numero, etc.), pero i es el estándar universal.

**Ejemplo: La Tabla de Multiplicar (for)**

Vamos a imprimir la tabla de multiplicar del 7

const numeroBase = 7;

console.log(`--- Tabla del ${numeroBase} ---`);

for (let i = 1; i <= 10; i++) {

let resultado = numeroBase \* i;

console.log(`${numeroBase} x ${i} = ${resultado}`);

}

// Salida:

// 7 x 1 = 7

// 7 x 2 = 14

// ...

**Ejemplo 2: El Buscador de Números Pares (while + if)**

Vamos a buscar el primer número par mayor a 100 generado aleatoriamente.

let encontrado = false;

let intentos = 0;

while (!encontrado) {

// Generamos un número al azar entre 90 y 110

let numeroRandom = Math.floor(Math.random() \* (110 - 90 + 1) + 90);

intentos++;

console.log(`Intento ${intentos}: Salió el ${numeroRandom}`);

if (numeroRandom > 100 && numeroRandom % 2 === 0) {

console.log(`¡Encontrado! El ${numeroRandom} es mayor a 100 y par.`);

encontrado = true; // Esto detendrá el ciclo

}

}

**Práctica Acompañada**

**Misión:** Calcular la suma de todos los números del 1 al 100. (1 + 2 + 3 + ... + 100).

**Paso a paso:**

1. **Abre tu VSC.**
2. **Crea una variable "acumuladora"**: Necesitamos una "caja" para guardar la suma total. let sumaTotal = 0;
3. **Configura el ciclo for**: Queremos un ciclo que se ejecute 100 veces, con un contador i que vaya del 1 al 100.
4. **Escribe la lógica del ciclo**: Dentro del for, toma el valor actual de sumaTotal y súmale el valor actual del contador i. Guarda el resultado de nuevo en sumaTotal.
5. **Muestra el resultado**: **Después** de que el ciclo termine, imprime el valor final de sumaTotal en la consola.

**Solución**

// 1. Crear la variable acumuladora

let sumaTotal = 0;

console.log(`Valor inicial de la suma: ${sumaTotal}`);

// 2. Configurar el ciclo for (del 1 al 100)

// Empezamos i en 1 y la condición es i <= 100

for (let i = 1; i <= 100; i++) {

// 3. Lógica del ciclo

// En cada vuelta, suma el valor de 'i' al acumulador

sumaTotal = sumaTotal + i;

// O en forma corta: sumaTotal += i;

// Opcional: ver el progreso

// console.log(`Sumando ${i}, la suma ahora es: ${sumaTotal}`);

}

// 4. Mostrar el resultado final

console.log(`La suma total de los números del 1 al 100 es: ${sumaTotal}`);

// Resultado esperado: 5050

**Práctica B Juego de Dados: ¡Evita el 1!**

El objetivo es sacar un 6. Empiezas con 3 vidas. Cada vez que lanzas y no sacas un 6, sigues jugando. Pero si sacas un 1, pierdes una vida. El juego termina si sacas un 6 (¡ganas!) o si te quedas sin vidas (¡pierdes!).

**Paso a paso:**

1. Crea las variables de estado:
   * Crea una variable let llamada vidas y asígnale 3.
   * Crea una variable let llamada dado y asígnale 0 (para guardar el resultado de cada lanzamiento).
   * Crea una variable let llamada ganoElJuego y asígnale false. Esta es una variable "bandera" (*flag*) que nos dirá si ganamos.
2. Configura el ciclo while: El juego debe continuar mientras la variable vidas sea mayor que 0.
3. Lanza el dado: Dentro del while, genera un número aleatorio del 1 al 6 y guárdalo en la variable dado.
   * Pista: dado = Math.floor(Math.random() \* 6) + 1;
4. Imprime el resultado: Muestra en consola qué sacaste en este lanzamiento. (ej. console.log(\Lanzaste un: ${dado}`);`)
5. Crea la lógica condicional (if...else if...else):
   * SI (if) el dado es igual a 6:
     + Imprime en consola "¡Sacaste un 6! ¡Ganaste!".
     + Cambia tu variable ganoElJuego a true.
     + Usa la palabra clave break; para salir del bucle while inmediatamente (ya que el juego terminó).
   * SI NO, PERO SI (else if) el dado es igual a 1:
     + Resta 1 a la variable vidas (Pista: vidas--).
     + Imprime cuántas vidas te quedan (ej. ¡Uy! Salió un 1. Pierdes una vida. Te quedan ${vidas} vidas.).
   * SI NO (else):
     + Imprime un mensaje como "Salió un [número]. No pasa nada, vuelves a lanzar."
6. Muestra el resultado final: Después de que el ciclo while termine (ya sea por break o porque vidas llegó a 0), añade un if final para comprobar si el jugador perdió.
   * if (ganoElJuego === false) { console.log('¡Game Over! Te quedaste sin vidas.'); }

**Troubleshooting**

**El Loop Infinito:**

* *Error:* for (let i = 0; i < 10; i--)
* *Problema:* Si empiezas en 0 y restas, i será -1, -2... siempre será menor a 10. Nunca para.
* *Solución:* Verifica siempre que tu lógica de actualización (i++ o i--) se acerque a romper la condición, no a alejarse.

**El error "Off-by-one" (Por uno):**

* *Error:* Quieres recorrer 5 veces y escribes for (let i = 0; i <= 5; i++).
* *Problema:* De 0 a 5 hay **6** números (0,1,2,3,4,5).
* *Solución:* Si empiezas en 0, usa <. Si empiezas en 1, usa <=.

**Scope de var vs let en ciclos:**

* Si usas var i = 0, la variable i sigue existiendo fuera del ciclo. Si usas let i = 0, la variable i muere al terminar el ciclo. **Siempre usa let en el for.**

**Ejercicio Independiente Aplicado**

**Misión: "Generador del 'FizzBuzz'"**

Este es un ejercicio de programación clásico que combina bucles, condicionales y el operador módulo (%).

**Instrucciones:**

1. Escribe un ciclo for que cuente del 1 al 100.
2. **Dentro** del ciclo, deberás aplicar las siguientes reglas para **cada número**:
   * **Si (if)** el número es divisible por 3 **Y** por 5 (ej. 15), imprime en consola: "FizzBuzz".
   * **SI NO, PERO SI (else if)** el número es solo divisible por 3 (ej. 9), imprime en consola: "Fizz".
   * **SI NO, PERO SI (else if)** el número es solo divisible por 5 (ej. 10), imprime en consola: "Buzz".
   * **SI NO (else)** (para todos los demás números), imprime el número en sí (ej. 7).
3. **Pista:** Para saber si un número es divisible por otro, usa el operador módulo (%). numero % 3 === 0 significa que "el residuo de numero dividido por 3 es 0", o sea, es divisible.

**Criterios de Logro Esperados:**

* El ciclo for se ejecuta 100 veces.
* Se utiliza una estructura if...else if...else.
* El orden de las condiciones es correcto (la comprobación de "FizzBuzz" debe ir **primero**).
* La salida en consola es correcta (1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz, ...).

**¿Debería Saber Algo Más?**

**Ciclos Anidados (Nested Loops):**

Puedes poner un for dentro de otro for. Esto es común para recorrer matrices (tablas de Excel con filas y columnas).

* ***Advertencia****:* Si tienes un ciclo de 100 vueltas dentro de otro de 100 vueltas, harás 10,000 operaciones. Esto puede hacer lenta tu aplicación (Concepto de Big O Notation).

**Iteradores Modernos:**

En el futuro, cuando veamos **Arreglos (Arrays)**, dejaremos de usar tanto el for clásico (let i=0...) y pasaremos a usar versiones más modernas y limpias como for...of o métodos como .forEach() y .map(). Pero entender el for clásico es obligatorio para entender la lógica base.

Logotipo

El contenido generado por IA puede ser incorrecto.