JAVASCRIPT

CLASE 3

Material complementario

CICLOS/ITERACIONES

CODER HOUSE

JavaScript: ciclos

Ciclos: definición

Los ciclos o bucles son estructuras que nos permiten repetir un conjunto de instrucciones (bloque) una cantidad determinada de veces. La repetición (o iteración, como decimos en programación) es consecutiva, lo cual implica que los bloques del ciclo se interpretan uno seguido del otro. Este funcionamiento permite realizar acciones màs de una vez, y/o efectuar cálculos relacionados.

El funcionamiento del bucle está sujeto a una comparación, esto significa que los ciclos funcionan hasta que cierta condición evaluada es falsa (false). En función de cómo se construye dicha condición, podemos identificar a los ciclos en dos grupos:

- Ciclos por conteo: el bloque de código se repite un número específico de veces. Para ello, se emplea una variable numérica que aumenta (o disminuye) en cada ciclo, representando la cantidad de repeticiones realizadas. Una vez que dicha variable alcanza el valor de iteraciones deseado, el bucle por conteo interrumpe su ejecución. La estructura ***for*** es de este tipo.
- Ciclos condicionales: implican comparaciones de todo tipo. De la misma forma que construimos los condicionales ***if***, las estructuras de este tipo interpretarán el bloque si la comparación es verdadera, sólo que a diferencia de ***if***, las instrucciones se repiten hasta que el condicional se evalúa como falso. Las estructuras ***while*** y ***do...while*** son de este tipo.

Estructura for

La estructura for en JavaScript garantiza la repetición de un conjunto de instrucciones hasta que cierta variable numérica tenga un valor específico. Analicemos a continuación su forma de declaración:

```
for (desde ; hasta ; actualización) {
    alert(i);
}
```

Podemos decir que el condicional ***for*** está compuesto por tres partes:

- **Desde:** implica el valor inicial de la variable, el cual determina desde qué números se empiezan a contar los ciclos. En esta sección, se declara una variable y se le asigna un valor que irá aumentando (o disminuyendo) en cada ciclo, según la actualización. Un tipo de declaración de desde es ***let i = 0*** o ***let index = 0.***
- **Hasta:** siendo un bucle de conteo, la sección hasta determina la comparaciòn que limita la iteraciòn. Aquí comparamos la variable declarada en desde con un valor que representa la "cantidad de repeticiones"; si la comparaciòn resulta falsa, el ***for*** finaliza, lo que implica que el bloque no vuelve a repetirse. Un ejemplo de declaraciòn es ***i < 10*** o ***index < 10***.
- **Actualización:** la última parte de la comparaciòn permite establecer cómo se cuenta en cada ciclo. Cuando termina una repetición, la variable de conteo va a cambiar teniendo en cuenta cómo está construida la actualización. El caso más común de actualización es ***i++*** o ***index++***, lo que quiere decir que en cada repetición el valor de la variable de conteo aumenta en uno (i++ es lo mismo que escribir ***i = i + 1***).

En resumen, con ***desde*** establecemos desde (valga la redundancia) qué valor empezamos a contar las repeticiones, con ***hasta*** definimos el número máximo de iteraciones, y con la ***actualización*** establecemos cómo vamos a realizar la modificación de la variable de conteo. Veamos ahora un ejemplo de uso de ***for*** para contar hasta 10:

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

Aquí el ***hasta*** inicia en uno , el *desde* va hasta 10 inclusive, porque la comparación es menor o igual, y la actualización se realiza de uno en uno; cuando i valga 11, lo cual ocurrirá al concluir la iteración 10, la estructura for deja de funcionar.

Ahora bien, es común iniciar la cuenta de los ciclos de conteo desde cero (0). Para ese caso, contar desde 0 a 9 implicaría la siguiente declaración:

```
for (let i = 0; i < 10; i++) {
    alert(i);
}
```

Uso de la estructura for

La estructura ***for*** tiene distintos casos de aplicación en la programación, uno de ellos, dada su característica de bucle de conteo, es la posibilidad de realizar operaciones incrementales o decrementales, valiéndose de la variable de conteo (que desde ahora llamaremos ***índice***). Veamos ahora un ejemplo de cómo podemos usar ***for*** para calcular la tabla de multiplicar de un número ingresado por el usuario:

```
// Solicitamos un valor al usuario
let ingresarNumero = parseInt(prompt("Ingresar Numero"));
// En cada repetición, calculamos el número ingresado x el número de repetición (i)
for (let i = 1; i <= 10; i++) {
    let resultado = ingresarNumero * i ;
    alert(ingresarNumero +" X "+ i +" = "+ resultado);
}
```

Por otro lado, recordemos que los bucles son consecutivos, y que en el índice se almacena el número de la iteración, lo cual convierte a ***for*** en un candidato para asignación ordinal, es decir, utilizar la variable de conteo para definir un orden en las operaciones. Veamos ahora cómo podemos usar ***for*** para informar veinte turnos, uno por cada nombre ingresado:

```
for (let i = 1; i <= 20; i++) {
    // En cada repetición solicitamos un nombre.
     let ingresarNombre = prompt("Ingresar nombre");
    // Informamos el turno asignado usando el número de repetición (i).
     alert(" Turno  N° "+i+" Nombre: "+ingresarNombre);
}
```

Break y continue

Cuando utilizamos un ciclo, puede surgir la necesidad de interrumpir o saltar una iteración en respuesta a la evaluación de cierto cálculo utilizando un condicional, para estos casos tenemos la sentencia break y continue respectivamente.

Veamos ahora un ejemplo sobre cómo podemos usar ***break*** para cancelar las repeticiones restantes (interrumpir) del ***for***, cuando el índice es igual a cinco:

```
for (let i = 1; i <= 10; i++) {
    //Si la variable i es igual 5 interrumpo el for.
     if(i == 5){
         break;
     }
     alert(i);
}
```

Si usamos ***continue*** en lugar de ***break***, las instrucciones restantes no se interpretan, y se pasa a la siguiente repetición, razón por la cual se dice coloquialmente que se "salta" un ciclo cuando se cumple la condición asociada al ***continue***.

```
for (let i = 1; i <= 10; i++) {
    //Si la variable i es 5, no se interpreta la repetición
     if(i == 5){
         continue;
     }
     alert(i);
}
```

Estas sentencias nos permiten establecer un control más específico sobre las variables manejadas dentro del bucle.

Estructura while

Cuando existe la necesidad de utilizar un bucle, pero su repetición está condicionada por el valor de una o más variables no numéricas, la estructura ***while*** es más apropiada.

Mientras la condiciòn se evalúe como verdadera, se repetirá el bloque de instrucciones definido (cabe aclarar que hay que prestar especial cuidado al momento de codificar el condicional). Cuando usamos ***while,*** asumimos que en algún momento la repetición va a finalizar; si la comparación no se realiza adecuadamente, como se visualiza en el ejemplo siguiente, podemos generar el llamado "bucle infinito", fenómeno que implica problemas en la ejecución de nuestro programa, ya que nunca finaliza la repetición del bloque, "colgando" la aplicación:

```
let repetir = true;
while(repetir){
    console.log("Al infinito y...¡Más allá!");
}
```

Entonces, cuando usamos ***while*** sabemos que el valor de la variable comparada va

```
7818 Al infinito y...¡Más allá!
```

cambiar en algún momento, de forma tal que el ciclo se interrumpe. Analicemos dicho funcionamiento desde un ejemplo, imaginando que tenemos que solicitar un valor al

usuario hasta que tipee escape (“ESC”); así, el código en cuestión podría ser:

```
let entrada = prompt("Ingresar un dato");
//Repetimos con While hasta que el usuario ingresa "ESC"
while(entrada != "ESC" ){
    alert("El usuario ingresó "+ entrada);
   //Volvemos a solicitar un dato. En la próxima iteración se evalúa si no es ESC.
    entrada = prompt("Ingresar otro dato");
}
```

Por cada ciclo se informa por ***alert*** al usuario la entrada capturada, y se solicita una nueva entrada. Antes de iniciar el próximo ciclo, nos preguntamos en la comparación si el usuario NO escribió “ESC”. Observemos que estamos usando el operador de comparación distinto (!=), y el único valor que puede dar falso tal comparación es que el usuario ingrese exactamente “ESC”; entonces, el bucle se repite mientra el usuario NO ingresa “ESC”.

Se dice que con ***while*** podemos hacer ninguna, una o más de una iteración. Para entender esta situación, focalizamos en la primera línea del ejemplo anterior: `let entrada = prompt("Ingresar un dato");` ¿Qué pasaría si el usuario ingresara “ESC” en la primera solicitud? En ese caso, la comparación del ***while*** es falta y no se repite ninguna vez.

En cambio, si el usuario ingresa “ESC” en la segunda solicitud: (`entrada = prompt("Ingresar otro dato");` al evaluar el condicional para la segunda repetición, el resultado es falso, por lo cual sólo se realizará una única interpretación del bloque.

En conclusión, la estructura ***while*** nos permite repetir un bloque siempre y cuando el valor del condicional evaluado sea verdadero; pero hay que tener presente que el valor previo de las variables comparadas puede determinar que el ciclo no se interprete ni una sola vez.

Estructura do...while

Anteriormente vimos que ***while*** puede no realizar repetición alguna si el condicional resulta falso (false) en el primer ciclo. A veces existe la necesidad de emplear una estructura que garantice al menos una interpretación del bloque de instrucciones de la estructura; en ese caso, usamos ***do...while***, que funciona de forma similar al ***while***, sólo que el condicional se evalúa al final de la iteración y no al comienzo, permitiendo ejecutar al menos una vez el conjunto de instrucciones:

```
let repetir = false;
do{
     console.log("¡Solo una vez!");
}while(repetir)
```

A pesar de que la variable posee el valor *false*, el mensaje por consola se visualiza una vez, ya que la comparación que determina la continuidad del ciclo está luego de la palabra reservada ***while.***

La estructura ***do...while*** puede simplificar el proceso de solicitud de entradas al usuario, ya que podemos solicitar las mismas, evaluarlas, y luego determinar si se realiza la repetición siguiente o no. Contrario al ***while,*** que al solicitar de entrada inicial se da lugar a la posibilidad de que el bucle no se ejecute jamás.

Veamos ahora un ejemplo de cómo usar ***do...while*** para asegurarnos de que el usuario siempre ingrese un número:

```
let numero = 0;
do{
   //Repetimos con do...while mientras el usuario ingresa un n°
    numero = prompt("Ingresar Número");
    console.log(numero);
//Si el parseo no resulta un número se interrumpe el bucle.
}while(parseInt(numero));
```

Se pide el primer número al usuario, garantizando que esa entrada se vaya a solicitar, y al terminar de interpretar todas las instrucciones del bloque se procede a evaluar el condicional si el valor ingresado se puede parsear se para a otro ciclo; caso contrario, el usuario ingreso un valor no numérico, y se interrumpe la carga de números

Estructura: switch

Como vimos, las estructuras ***while*** y ***do...while*** nos permiten controlar el envío de entradas por parte del usuario de forma apropiada. No obstante, cuando se tiene que evaluar el valor de una entrada más de una vez dentro del ciclo, en lugar de emplear un **if...else if** podemos usar la estructura condicional ***switch***, la cual tiene la siguiente forma de declaración:

```
switch(numero) {
 case 5:
    ...
   break;
 case 8:
    ...
   break;
 case 20:
    ...
   break;
 default:
    ...
   break;
}
```

"numero" es una variable que puede tener distintos valores, y por cada uno posible que queramos controlar, utilizamos la palabra reservada ***case,*** seguida del valor a evaluar. Si se cumple el caso, es decir, si la variable tiene ese valor, se interpreta el conjunto de

instrucciones asociadas a él, y el bloque a interpretar se codifica entre los dos puntos (:) y el break.
Únicamente es posible ingresar a un caso de las estructuras si, por ejemplo, el valor de la variable evaluado usando switch es igual a veinte, y sólo se ejecuta el bloque asociado al valor mencionado; luego, gracias al break, se sale de la estructura.

Si la variable evaluada no tiene ninguno de los valores identificados en case, se procede a interpretar el bloque en default como respuesta predeterminada cuando ninguno de los valores buscados se encuentra.

Un ejemplo aplicado de ***switch*** y ***while*** se identifica cuando tenemos la necesidad de controlar la entrada a evaluar, una vez que sabemos que no interrumpe el ciclo. Es decir, una vez que sabemos que la entrada no es "ESC", buscamos determinar qué valor tiene para asociar un conjunto de instrucciones de respuesta personalizadas para cada valor:

```
let entrada = prompt("Ingresar un nombre");
//Repetimos hasta que se ingresa "ESC"
while(entrada != "ESC" ){
   switch (entrada) {
       case "ANA":
            alert("HOLA ANA");
            break;
        case "JUAN":
            alert("HOLA JUAN");
            break;
       default:
           alert("¿QUIÉN SOS?")
           break;
   }
   entrada = prompt("Ingresar un nombre");
}
```

Combinando estructuras

En el ejemplo anterior, vimos cómo anidar un ***switch*** dentro de un ***while*** nos permite establecer un control en el programa cada vez más específico. Este es el objetivo al que apuntamos como programadores/as: establecer cómo podemos emplear las estructuras y elementos para dar solución a problemas cada vez más complejos.

Las combinaciones de estructuras pueden variar, y en algunos casos podemos decir que ciertas estructuras son más apropiadas que otras (como el caso de ***do...while*** sobre ***while*** para asegurar la primer iteración, o el empleo de ***for*** cuando nos importa determinar qué numero de repeticion estamos ejecutando). Pero es importante distinguir que en la programación una cosa es la resolución del problema, lo cual es prioritario, y otra es determinar cuál es el conjunto de estructuras y elementos más apropiadas para resolverlo, ya que no existe tal cosa como el algoritmo perfecto, sino distintas aproximaciones más o menos eficaces para solucionar un inconveniente.

En consecuencia, es tan importante el conocimiento en el lenguaje como el ejercicio y la experiencia en la resolución de distintas situaciones prácticas, empleando combinaciones de instrucciones, siempre estudiando su conveniencia en la aplicación que estamos programando.