

BUCLES O CICLOS DE REPETICIÓN

Abiertos y cerrados

¿Qué son los bucles de repetición?

- Cuando necesitamos realizar **acciones en forma repetida**, en vez de escribir varias veces un código, podemos recurrir a una **estructura de repetición**.
- Por ejemplo, si necesitamos que se ingresen 20 números en forma consecutiva, o si tenemos que ingresar las notas de una comisión de... no debemos escribir 20 veces o más el código.
- Hay **dos tipos de ciclos o bucles** de repetición:
 - ▣ Abiertos
 - ▣ Cerrados

Bucles cerrados: for

- Son bucles en los que **podemos predecir**, al escribir el código, la cantidad de vueltas, ciclos o repeticiones que dará el bucle.
- Dentro de este grupo encontramos el **bucle FOR** que presenta la siguiente estructura.

```
for(parámetros )  
    acciones a repetir
```

Bucles cerrados: for

- Son bucles en los que **podemos predecir**, al escribir el código, la cantidad de vueltas, ciclos o repeticiones que dará el bucle.
- Dentro de este grupo encontramos el **bucle FOR** que presenta la siguiente estructura.

```
for(parámetros) {  
    acciones a repetir  
}
```

Bucles cerrados: for

- Recibe **TRES** parámetros o argumentos, separados por **PUNTO Y COMA** (es la única estructura de programación en la que se usa el ; en vez de la , para separarlos).
-

la siguiente estructura.




```
for(parámetros ) {  
    acciones a repetir  
}
```

Parámetros del bucle for

```
for( valor de Condición Incremento o  
    inicio; de corte; decremento ) {  
    acciones a repetir  
}
```

Parámetros del bucle for

```
for( valor de Condición Incremento o  
    inicio; de corte; decremento ) {  
    acciones a repetir  
}
```




El primer argumento se trata del valor inicial de una **VARIABLE DE ITERACIÓN o REPETICIÓN** (la variable que va a controlar la cantidad de vueltas o repeticiones que dará el bucle). Ese valor será el que tendrá la variable **SOLO** en la **PRIMERA VUELTA**.

Parámetros del bucle for

```
for( valor de Condición Incremento o  
    inicio; de corte; decremento ) {  
    acciones a repetir  
}
```


Parámetros del bucle for

```
for( valor de Condición Incremento o  
    inicio; de corte; decremento ) {  
    acciones a repetir  
}
```




Es una condición que determina si el bucle debe dar otra vuelta o terminar. Si la condición resulta **TRUE**, entra al bucle y repite, si devuelve **FALSE**, lo corta inmediatamente sin hacer más nada (siguen las acciones que continúen por debajo de la llave de cierre del bucle).

Parámetros del bucle for

```
for( valor de Condición Incremento o  
    inicio; de corte; decremento ) {  
    acciones a repetir  
}
```

Parámetros del bucle for

```
for( valor de Condición Incremento o  
    inicio; de corte; decremento ) {  
    acciones a repetir  
}
```



Determina cómo cambia el valor de la variable de una vuelta a la siguiente. Puede incrementar o decrementar su valor dependiendo lo que convenga para alcanzar el valor de corte del bucle. Lo hacemos utilizando operadores como ++, --, +=, -=, etc.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

PRIMERA VUELTA O ITERACIÓN

Funcionamiento del bucle for

```
for( var i=1; i<5; i++ ) {  
    acciones a repetir  
}
```

- Se declara una **variable que controla la iteración** o se puede usar una ya existente.
- Se asigna el **valor** que tendrá la variable **SOLO en la primera vuelta**. Es su valor inicial.

PRIMERA VUELTA O ITERACIÓN

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

PRIMERA VUELTA O ITERACIÓN

- Luego **se evalúa la condición de corte** del bucle.
- Si resulta **true, entra al bucle** para hacer por primera vez las acciones a repetir. Si resultara **false, no entra ni una sola vez** y continúa con el código a continuación de la llave de cierre.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```

- Se “lee” qué **incremento** o **decremento** se aplicará a la variable pero **NO SE APLICA EN ESTE MOMENTO** (sino el primer parámetro no sería el valor inicial y se estaría evaluando la condición con un valor que inmediatamente cambia).

PRIMERA VUELTA O ITERACIÓN

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

- Se realizan todas las acciones que se encuentren entre las llaves...

PRIMERA VUELTA O ITERACIÓN

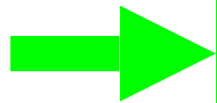
Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

- Una vez que se terminaron las acciones de la primera vuelta **SE APLICA EL INCREMENTO O DECREMENTO** para preparar la variable para entrar o no en la siguiente vuelta.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

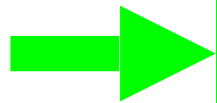


Recién al llegar al final se aplica el incremento `i++`

- Una vez que se terminaron las acciones de la primera vuelta **SE APLICA EL INCREMENTO O DECREMENTO** para preparar la variable para entrar o no en la siguiente vuelta.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```



Recién al llegar al final se aplica el incremento `i++`

FIN DE LA PRIMERA VUELTA

- Una vez que se terminaron las acciones de la primera vuelta **SE APLICA EL INCREMENTO O DECREMENTO** para preparar la variable para entrar o no en la siguiente vuelta.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

- **ANTES DE EMPEZAR UNA ACLARACIÓN... el primer parámetro NO SE EJECUTA MÁS**, ya que es el valor SOLO para la primera vuelta.
- Directamente se evalúa la condición de corte y si resulta true...

SEGUNDA VUELTA

Funcionamiento del bucle for

```
for(var X i=1; i<5; i++) {  
    acciones a repetir  
}
```

SEGUNDA VUELTA

- **ANTES DE EMPEZAR UNA ACLARACIÓN... el primer parámetro NO SE EJECUTA MÁS**, ya que es el valor SOLO para la primera vuelta.
- Directamente se evalúa la condición de corte y si resulta true...

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

□ Se realizan las acciones...

SEGUNDA VUELTA

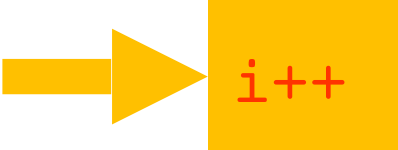
Funcionamiento del bucle for

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```

- Y, al terminarlal, se incrementa el valor de la variable para prepararla para la siguiente vuelta.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```



- Y, al terminarlas, se incrementa el valor de la variable para prepararla para la siguiente vuelta.

FIN DE LA SEGUNDA VUELTA

Funcionamiento del bucle for

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```

- A partir de la segunda vuelta en adelante **SIEMPRE** se realizan los mismos pasos...
- Primero se evalúa la condición de corte y si resulta true...

TERCERA VUELTA Y +

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

□ Se realizan las acciones...

TERCERA VUELTA Y +

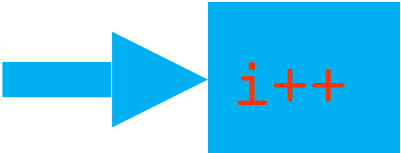
Funcionamiento del bucle for

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```

- Y, al terminarlal, se incrementa el valor de la variable para prepararla para la siguiente vuelta.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```



- Y, al terminarlas, se incrementa el valor de la variable para prepararla para la siguiente vuelta.

FIN DE LA VUELTA

Funcionamiento del bucle for



¿Hasta cuándo hace siempre lo mismo?

Funcionamiento del bucle for

FALSE

```
for(var i=1; i<5; i++) {  
    acciones a repetir  
}
```

- Hasta que al comenzar una nueva vuelta **la condición de corte resulte false** y el bucle termina, NO ENTRA MÁS.

Funcionamiento del bucle for

```
for(var i=1; i<5; i++ ){  
    acciones a repetir  
}
```

```
alert("El bucle  
terminó");
```

- Hasta que al comenzar una nueva vuelta **la condición de corte resulte false** y el bucle termina, NO ENTRA MÁS.
- Continúa con el código que se encuentre a continuación del bucle...

Analicemos la cantidad de vueltas del ejemplo

```
for(var i=1; i<5; i++ ){  
    console.log("hola");  
}
```

- Primera vuelta:
 - ▣ Valor de i: 1
 - ▣ Condición: true
- Segunda vuelta:
 - ▣ Valor de i: 2
 - ▣ Condición: true
- Tercera vuelta:
 - ▣ Valor de i: 3
 - ▣ Condición: true
- Cuarta vuelta:
 - ▣ Valor de i: 4
 - ▣ Condición: true
- Quinta vuelta:
 - ▣ Valor de i: 5
 - ▣ Condición: FALSE → TERMINA

Analicemos la cantidad de vueltas del ejemplo

```
for(var i=1; i<5; i++ ){  
    console.log("hola");  
}
```

¿Cuántos holas veremos en consola?

- Primera vuelta:
 - ▣ Valor de i: 1
 - ▣ Condición: true
- Segunda vuelta:
 - ▣ Valor de i: 2
 - ▣ Condición: true
- Tercera vuelta:
 - ▣ Valor de i: 3
 - ▣ Condición: true
- Cuarta vuelta:
 - ▣ Valor de i: 4
 - ▣ Condición: true
- Quinta vuelta:
 - ▣ Valor de i: 5
 - ▣ Condición: FALSE → TERMINA

Analicemos la cantidad de vueltas del ejemplo

```
for(var i=1; i<5; i++ ){  
    console.log("hola");  
}
```

¿Cuántos holas veremos en consola?

hola
hola
hola
hola

- Primera vuelta:
 - ▣ Valor de i: 1
 - ▣ Condición: true
- Segunda vuelta:
 - ▣ Valor de i: 2
 - ▣ Condición: true
- Tercera vuelta:
 - ▣ Valor de i: 3
 - ▣ Condición: true
- Cuarta vuelta:
 - ▣ Valor de i: 4
 - ▣ Condición: true
- Quinta vuelta:
 - ▣ Valor de i: 5
 - ▣ Condición: FALSE → TERMINA

Bucles abiertos: while


- Son bucles en los que **NO podemos predecir**, al escribir el código, la cantidad de vueltas, ciclos o repeticiones que dará el bucle.
- Dentro de este grupo encontramos el **bucle WHILE** que presenta la siguiente estructura.

```
while( condición de corte ) {  
    acciones a repetir  
}
```

Bucles abiertos: while

- Son bucles en los que **NO podemos predecir**, al escribir el código, o repeticiones que dará el bucle. **Solo recibe UN parámetro**
- Dentro de este grupo encontramos el **bucle WHILE** que presenta la siguiente estructura.

```
while( condición de corte ) {  
    acciones a repetir  
}
```




Funcionamiento del bucle while

```
var i=1;
while( i<5 ) {
    acciones a repetir
}
```

- Se evalúa **directamente la condición de corte**.
- Si resulta **true**, **entra al bucle** para hacer las acciones. Si resultara **false**, **no entra ni una sola vez** y continúa con el código a continuación de la llave de cierre.

Funcionamiento del bucle while

```
var i=1;  
while(i<5){  
    acciones a repetir  
}
```



- ¿Cuál es el valor de la variable *i* y cómo resulta la condición en la primera vuelta?
- ¿Y en la segunda?
- ¿Y en la tercera?
- El **peligro** más grande de este tipo de ciclos es el **bucle infinito**.

Funcionamiento del bucle while

```
var i=1;
while(i<5){
    acciones a repetir

    i++;
}
```

- Para evitarlo **SIEMPRE** se debe **incluir dentro de las acciones** del bucle la posibilidad que la variable **llegue al valor de corte**.
- ¿Cuántas vueltas dará este bucle?
- Pero entonces... está funcionando como **cerrado!!!**.

Funcionamiento del bucle while

```
var i=1;
while(i<5){
    accio
    i++;
}
```

Los bucles abiertos pueden trabajar tanto como abiertos, como cerrados, de acuerdo a cómo se plantea su estructura. En cambio los cerrados, nunca pueden trabajar como abiertos.

- Para evitarlo **SIEMPRE** se debe **incluir dentro de las acciones** la posibilidad que la **condición** se al valor de **verdadero** as dará este
- Pero entonces... está funcionando como **cerrado!!!**.

Funcionamiento del bucle while

```
while(i<5){  
    acciones a repetir  
    i+=prompt("Ingrese un  
    número");  
}
```

- ¿Y ahora, cuántas vueltas dará el bucle?
- **No podemos predecir** si serán 2, 100 o 1 vuelta ya que depende del ingreso del usuario.
- Ahora funciona como **bucle abierto**.

Bucles abiertos: do - while

- Otro bucle abierto es el bucle do-while, es similar al anterior pero **la condición se incluye al final** de la estructura en vez de al comienzo.

Bucles abiertos: do - while

- Otro bucle abierto es el bucle do-while, es similar al anterior pero **la condición se incluye al final** de la estructura en vez de al comienzo.

do

Bucles abiertos: do - while

- Otro bucle abierto es el bucle do-while, es similar al anterior pero **la condición se incluye al final** de la estructura en vez de al comienzo.

```
do {
```

```
    acciones a repetir que  
    Incluyan la posibilidad de  
    alcanzar el valor de corte
```

```
}
```

Bucles abiertos: do - while

- Otro bucle abierto es el bucle do-while, es similar al anterior pero **la condición se incluye al final** de la estructura en vez de al comienzo.

```
do {
```

```
    acciones a repetir que  
    Incluyan la posibilidad de  
    alcanzar el valor de corte
```

```
} while
```

Bucles abiertos: do - while

- Otro bucle abierto es el bucle do-while, es similar al anterior pero **la condición se incluye al final** de la estructura en vez de al comienzo.

```
do {
```

```
    acciones a repetir que  
    Incluyan la posibilidad de  
    alcanzar el valor de corte
```

```
} while( condición de corte );
```

Funcionamiento del bucle do-while

```
do{  
    acciones a repetir  
    i++; (o i+=prompt(...))  
} while(i<5);
```

- Se **realizan las acciones** del bucle.
- Luego se **evalúa la condición de corte**.
- Si resulta **true**, **vuelve al do** para realizar una nueva vuelta. Si resultara **false**, **termina el bucle** y continúa con el código a continuación.

Diferencia entre while y do-while

```
var i=5;

while(i<5){

    acciones a repetir

}
```

```
var i=5;

do{

    acciones a repetir

} while(i<5);
```

- ¿Cuántas vueltas da cada bucle?
- La única diferencia es que do-while **SIEMPRE** realiza **POR LO MENOS UNA VEZ** las acciones del bucle (cuando la condición no se cumple).

El objeto Math

- Posee **métodos y propiedades** creados por **constantes y funciones matemáticas**: PI, seno, logaritmo, raíz cuadrada, etc.
- De estos **nos interesan algunos métodos** que nos pueden servir en distintos proyectos.
- Se **acceden** mediante el **objeto Math** (con mayúscula) por medio de notación de punto:

Math.método()

Números aleatorios

- Para generar **números al azar** tenemos el método **random()** con la siguiente sintaxis:

`Math.random()`

- Dentro de los **paréntesis NO VA NADA**.
- Genera números flotantes **entre 0 (incluido) y 1 (NO incluido)**. Nunca entrega el límite superior (1). Ejemplos:

0.32383509807524513

0.7364674246188696

0.4841236188168745

0.8475775955375058

Rangos de números aleatorios

- Si **multiplicamos por un valor** el random obtenemos **números aleatorios entre 0 y ese valor**:

```
Math.random( ) * 10
```

0.32383509807524513 → 3.2383509807524513

0.7364674246188696 → 7.364674246188696

0.4841236188168745 → 4.841236188168745

0.8475775955375058 → 8.475775955375058

Rangos de números aleatorios

- Si necesitamos **un rango más específico** podemos aplicar la **siguiente fórmula**:

`Math.random()` * (max - min) + min

- Obtendremos números aleatorios entre el **número mínimo incluido** y el **máximo excluido**.
- Y si queremos **números enteros** e, incluso, que pueda entregarse el máximo del rango, tenemos que combinarlo con los **métodos de redondeo de decimales** del objeto Math: **round()**, **floor()** y **ceil()**.

Redondeos de decimales

- Presentan la siguiente sintaxis:

`Math.metodo(número o variable con decimales)`

- Y funcionan de la siguiente manera:

- `round()` → x,5 o mayor: devuelve el **entero superior**
→ menor a x,5: devuelve el **entero inferior**

- `floor()` → devuelve **SIEMPRE** el **entero inferior**

- `ceil()` → devuelve **SIEMPRE** el **entero superior**

- Combinando:

`Math.round(Math.random() * 10)` [1 a 10 inclusive]