

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: CICLOS DE CONTROL DE FLUJO.
- EXERCISE 2: IMPLEMENTANDO CONTROL DE FLUJO MEDIANTE UN SWITCH.
- EXERCISE 3: DO WHILE.

EXERCISE 1: CICLOS DE CONTROL DE FLUJO

Tal como se había mencionado en la lectura “Control de Flujo en JavaScript”, se puede implementar este concepto mediante bucles o ciclos, que nos permiten ejecutar código bajo cierto criterio o únicamente si satisface ciertas condiciones. Uno de ellos es el condicional If, que en castellano sería: “Si...” (por ejemplo, “si... se cumple ciertos requisitos... entonces ejecuta el código”).

La estructura de un **if** es la siguiente:

```
1 var condicion = true;
2
3 // if
4 if (condicion) { // Si se cumple la condición ...
5
6     // Se ejecuta este bloque de código.
7
8 }
```

Por ejemplo, supongamos que queremos ejecutar una función, solamente cuando el **input** de un formulario contenga un **String** o un número específico. Nuestro input se puede asemejar a lo siguiente:

```
1 <form>
2     <h2>Ingresa cualquier valor:</h2><br>
3     <input type="text" id="input" name="input"
4     onkeyup="eval(value)"><br>
5 </form>
```

Mediante éste, invocamos a la función **eval()** cada vez que dejamos de presionar una tecla, y por parámetro le pasamos el valor del input que queremos evaluar. Nuestra función será la siguiente:

```
1 function eval(valor) {
```

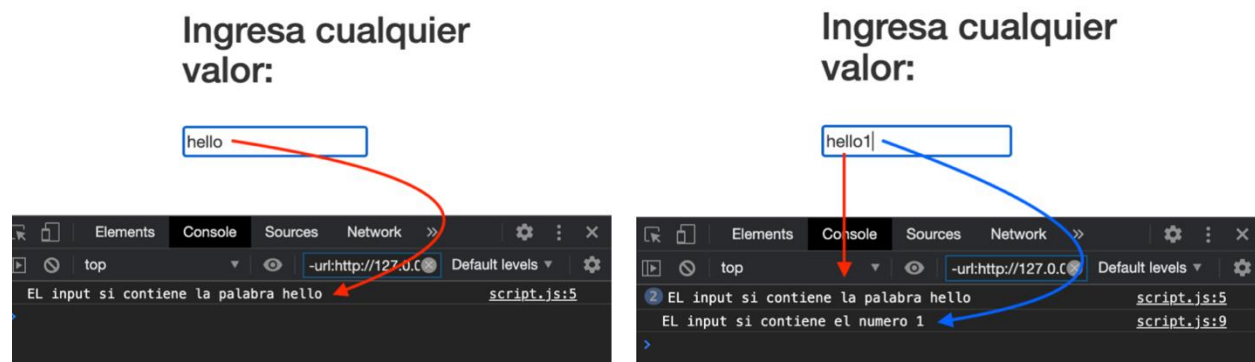
```

2   if (valor.includes("hello")) {
3       console.log("EL input si contiene la palabra hello")
4   }
5
6   if(valor.match(1)){
7       console.log("EL input si contiene el numero 1")
8   }
9 }

```

Ésta establece que si el valor del input incluye el **String "hello"**, entonces por consola visualizaremos el mensaje: **"el input si contiene la palabra hello"**, o si el valor del input contiene el número uno, también veremos el mensaje que indica el segundo **if**.

En nuestro navegador, el resultado es el siguiente:



Al escribir la palabra **hello** en el **input**, vemos que se satisface la primera condición dentro de nuestra función **eval()**, haciendo que en la consola recibamos el primer mensaje, y que al ingresar un número 1, recibamos otro mensaje similar, dado que estos valores cumplieron con las condiciones de los 2 ciclos **If**.

Estos ciclos se pueden enriquecer aún más cuando incorporamos operadores de comparación, dentro del parámetro de los bucles. Los operadores de comparación y de igualdad que se pueden utilizar en JavaScript, son los siguientes:

- **==** igual a...
- **===** igual valor e igual tipo.
- **!=** no igual.



- **!=** no igual valor ni tipo.
- **>** mayor que...
- **<** menor que...
- **>=** mayor o igual que...
- **<=** menor o igual que...

Se puede observar el funcionamiento de estos operadores de comparación dentro de la condición de un **if**, como muestra el siguiente **input** y función:

```
1 <form>
2     <h2> Ingresa un número:</h2><br>
3     <input type="number" id="input" name="input"
4     onkeyup="eval(value)"><br>
5     <h3 id="result"></h3> <!-- Aquí mostraremos un mensaje descriptivo
6     del input -->
7 </form>
```

```
1 var resultado = "esta variable indica si es positivo, cero o negativo.";
2
3 function eval(valor) {
4     // si el valor es igual a 0, entonces la variable resultado será
5     "cero".
6     if (valor == 0) { resultado = "cero" }
7
8     // si el valor es mayor a cero, entonces la variable resultado será
9     "positivo".
10    if (valor > 0) { resultado = "positivo" }
11
12    // si el valor es menor a cero, entonces la variable resultado será
13    "negativo".
14    if (valor < 0) { resultado = "negativo" }
15
16    document.getElementById("result").innerHTML = `El input '${valor}' es
17    ${resultado}.`;
18 }
```

El resultado en nuestro navegador demuestra lo siguiente. Al ingresar el número uno, el mensaje indica qué su número es positivo. Si el input es un cero, se indica que es un cero; pero si es el número negativo, se indica aquello:



Ingresa un
número:

El input '1' es
positivo.

Ingresa un
número:

El input '0' es cero.

Ingresa un
número:

El input '-1' es
negativo.

Se puede simplificar el *flow control* (control de flujo) de nuestra función `eval()`, usando un `if else`. Este ciclo establece que, si se cumple una condición, se ejecuta cierto código, pero si no se cumple, se efectúa por defecto otro bloque de código.

La estructura de un `if else` es la siguiente:

```
1 // if else
2 if (condicion) { // Si se cumple la condición ...
3
4     // Se ejecuta este bloque de código.
5
6 } else { // o si no se cumple ...
7
8     // Se ejecuta este otro bloque de código.
9
10 }
```

Utilizaremos el mismo `input`, pero modificaremos la función `eval()`, para incorporar el `if else`. El resultado es el siguiente:

```
1 var resultado = "esta variable indica si es positivo, cero o negativo.";
2
3 function eval(valor) {
4     if (valor >= 0) {
5         // si el valor es mayor o igual a cero, entonces la variable
6 resultado será "positivo".
7         resultado = "positivo"
```



```
8     } else {
9         // O sino, entonces la variable resultado será "negativo".
10        resultado = "negativo"
11    }
12    document.getElementById("result").innerHTML = `El input '${valor}' es
13    ${resultado}.`;
14 }
```

En nuestro navegador se aprecia el siguiente comportamiento:

**Ingresa un
número:**

**El input '1' es
positivo.**

**Ingresa un
número:**

**El input '-1' es
negativo.**

¿Existe algún recurso para poder repetir código sobre la base de una condición?: sí, para lograrlo se emplea el ciclo **for**, el cual es otro mecanismo de control de flujo.

La estructura de un **for** es la siguiente:

```
1 var cantidadIteraciones = 7;
2
3 // for
4 for (let i = 0; i < cantidadIteraciones; i++) {
5     // código que deseamos repetir
6
7 }
```

Podemos probar el comportamiento de este ciclo mediante el siguiente HTML, y función de JavaScript:

```
1 <form>
2     <h2> Prueba de ciclo for:</h2><br>
3     <label for="Valor">Valor por repetir:</label>
4     <input type="text" id="valor" name="valor"><br>
```



```

5     <label for="Veces">Veces por repetir:</label>
6     <input type="number" id="veces" name="veces"
7     onkeyup="repite()" "><br>
8 </form>
  
```

La función **repite()**:

```

1 function repite() {
2     // Obtenemos los valores desde el input:
3     var valor = document.getElementById('valor').value;
4     var veces = document.getElementById('veces').value;
5
6     // Repetiremos un valor por la cantidad de veces que indica el
7     usuario:
8     for (let i = 0; i < veces; i++) {
9         console.log(` (${i}) ${valor}`);
10    }
11 }
  
```

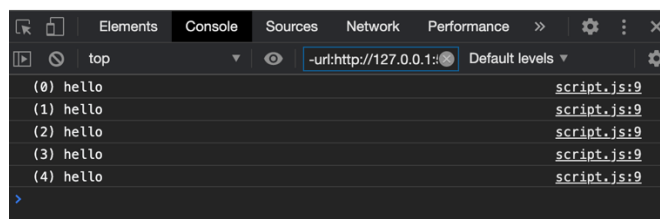
Ésta usa los valores de los 2 **input** de nuestro HTML, en el **ciclo for**. Este bucle o ciclo, establece que mientras un iterador **i** incremente su valor hasta ser del mismo que la variable **veces**, entonces se mostrará por consola el iterador y a la vez, el valor de la variable **“valor”**.

El resultado de esta función en nuestro navegador, muestra lo siguiente:

Prueba de ciclo for:

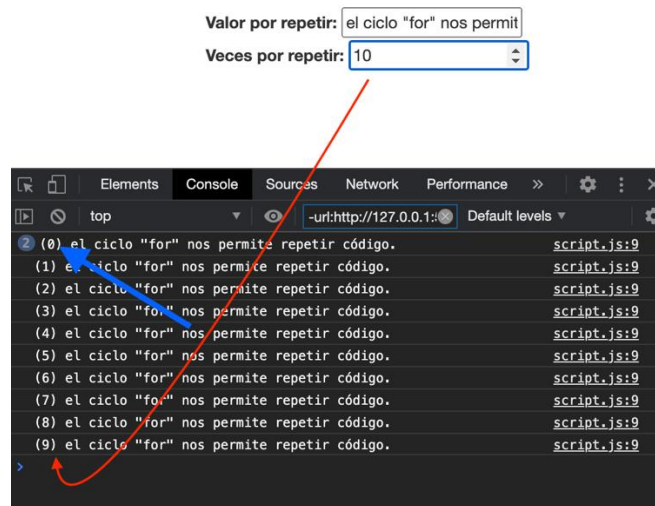
Valor por repetir:

Veces por repetir:



Cómo se puede observar, una vez que ingresamos un texto que queremos repetir, e indicamos la cantidad de veces que lo haremos, la consola lo mostrará de esa forma.

Prueba de ciclo for:



Si nos fijamos en la consola, quizás pudiésemos identificar un problema, siendo éste que las iteraciones no terminan en **10** (tal como indica la flecha roja) como nosotros queríamos. Esto se debe a que, dentro del ciclo **for**, establecimos que la variable que se va a iterar empieza con un valor de **cero**, por lo que en la consola podemos apreciar que la primera impresión parte por el número cero, y termina en la iteración número nueve, aunque esta es la décima vez que se imprime. Si quisiéramos “arreglar” esto, solamente bastaría con hacer el siguiente ajuste:

```
1   for (let i = 1; i < veces+1; i++) { // El iterador "i" parte en uno.
2       console.log(`(${i}) ${valor}`);
3   }
```

El cual consiste en establecer que el valor inicial de **i** será uno, y que se repetirá hasta la cantidad de veces que establecimos **+1**. Al indicar esto, pasaremos por alto el valor cero, por lo que nos faltaba agregarle uno más a la cantidad total, es decir: **veces + 1**.

De esta manera comprobamos el funcionamiento de dos ciclos muy importantes dentro del Control de Flujo en el lenguaje de JavaScript.

EXERCISE 2: IMPLEMENTANDO CONTROL DE FLUJO MEDIANTE UN SWITCH

Hasta el momento hemos revisado tres elementos, que nos ayudan a controlar el flujo de nuestro código JS, los cuales son: `if`, `if else` y el ciclo `for`. Éstos son recursos muy útiles, dado que nos permiten evaluar si el `input` satisface ciertas condiciones; pero si consideramos que los programas son escalables, y que pueden incrementar su funcionalidad día tras día, ¿tendremos que repetir los ciclos cada vez que se agreguen funcionalidades, o que se ingrese otra condición por evaluar?: la respuesta en realidad consiste en analizar otro mecanismo de control de flujo, que tiene la capacidad de evaluar muchas condiciones a la vez, y este es el `switch case`. En español, `Switch` se puede traducir a interruptor, y si pensamos en una casa que tiene más de uno para encender las luces en distintas habitaciones, nos asemejaremos al `switch` que nos permite controlar el código bajo distintas circunstancias.

La estructura de un `switch` es la siguiente:

```
1 // switch
2 switch (key) {
3     case value:
4         // Si el "value" equivale al "key", entonces se ejecutará este
5         bloque de código.
6
7         break; // Aquí este bloque de código para.
8
9
10    default: // aquí se efectúa código por defecto si el "value" no
11    equivale al "key".
12        break;
13 }
```

¿Cómo funciona? éste va comparando valores de caso (`case value`), con una llave (`key`). En sí, el `switch` compara el valor de la llave con los valores de caso; si el valor de caso es igual a la llave, entonces se efectúa el bloque de código bajo él, hasta llegar a la palabra clave `break`, dónde dicho bloque de código deja de efectuarse; pero, si ningún valor de caso equivale al valor de la llave, entonces la ejecución del código se va a la palabra clave `default`, dónde podemos indicar un código que se puede efectuar "por



defecto". Una vez ejecutados los bloques de código que satisfagan el requerimiento de la llave, el **switch** se detiene.

Ahora, vamos a analizar su comportamiento al momento de ser aplicado a un código HTML. A continuación, se podrá ver un formulario HTML, y posteriormente nuestra función cambia, pues incluye un **switch**:

```
1 <form>
2     <h2> Prueba de Swtich Case:</h2><br>
3     <label for="opciones">Seleccione una opción:</label>
4     <select name="opciones" onchange="cambia(value)">
5         <option value="1">Opción 1</option>
6         <option value="2">Opción 2</option>
7         <option value="3">Opción 3</option>
8     </select>
9 </form>
```

La función **cambia()**:

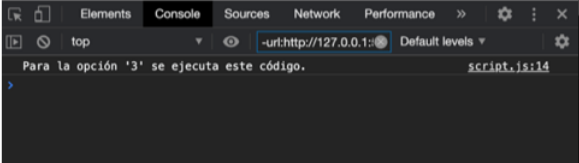
```
1 function cambia(valor) {
2
3     switch (valor) {
4         case '1':
5             console.log(`Para la opción '${valor}' se ejecuta este
6 código.`);
7             break;
8
9         case '2':
10            console.log(`Para la opción '${valor}' se ejecuta este
11 código.`);
12            break;
13
14         case '3':
15            console.log(`Para la opción '${valor}' se ejecuta este
16 código.`);
17            break;
18
19         default:
20            break;
21     }
22 }
```

Esta función recibe por parámetro el valor de la opción que seleccionamos en nuestro HTML, y mediante un **switch**, evalúa qué bloque de código efectuar en base a si el caso cobra el mismo valor que la llave **“valor”**.

En nuestro navegador, el switch se comporta de la siguiente manera:

Prueba de Swtich Case:

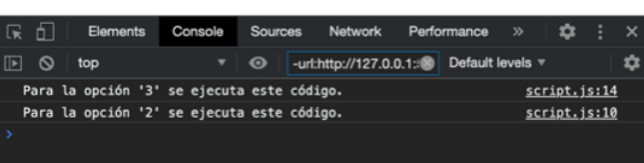
Seleccione una opción: Opción 3 ▼



Para la opción '3' se ejecuta este código. script.js:14

Prueba de Swtich Case:

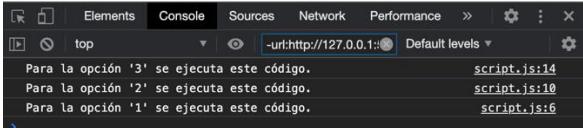
Seleccione una opción: Opción 2 ▼



Para la opción '3' se ejecuta este código. script.js:14
Para la opción '2' se ejecuta este código. script.js:10

Prueba de Swtich Case:

Seleccione una opción: Opción 1 ▼



Para la opción '3' se ejecuta este código. script.js:14
Para la opción '2' se ejecuta este código. script.js:10
Para la opción '1' se ejecuta este código. script.js:6

Como se puede observar, cada vez que seleccionamos una opción distinta, nuestro **switch** controla el flujo de la ejecución del código, y mediante la consola, muestra el que corresponde a cada opción. Tanto el **switch**, como los bucles que hemos analizado, son recursos potentemente útiles que nos permitirán controlar el flujo de nuestro código en JavaScript.

EXERCISE 3: DO WHILE

Todos los diferentes ciclos que hemos utilizado hasta el momento nos permiten dirigir el flujo de nuestro código de una manera distinta, y cada uno tiene sus particularidades. Continuaremos aprendiendo sobre el ciclo **do while**, que en castellano podríamos entender como “haz mientras”, y como lo indica su nombre, éste va a realizar de manera repetitiva una instrucción determinada, mientras se cumpla una condición.

Esto quizás nos puede hacer pensar que el ciclo **do while** realiza lo mismo que el ciclo **for**, pero no es el caso, pues el primero si o si va a ejecutar su código una vez, independiente de si se cumple la condición o no. Esto se debe a que verifica la condición *al final* de cada ejecución; mientras que, en el segundo, sucede lo opuesto, pues éste evalúa la condición antes de ejecutarse.

La sintaxis del ciclo **do while** parte con la palabra clave “do”, unas llaves donde colocaremos la instrucción que queremos repetir, y luego la palabra “**while**” que evaluará nuestra condición entre paréntesis.

```
1 do {  
2     // instrucción  
3 }  
4 while (condición);
```

UNAS PRECAUCIONES

Si usas una variable en la condición, debes inicializarla antes del ciclo, e incrementarla dentro de éste. De lo contrario, nunca terminará, provocando que se bloquee tu navegador. Si la condición es siempre verdadera, el ciclo nunca terminará. Esto también bloqueará tu navegador, así que procura tener en cuenta estos puntos antes de utilizar el bucle.

Como ya se han mencionado algunas precauciones que debemos tener, pondremos en práctica este ciclo con un ejemplo. Vamos a iterar el ciclo **do while** unas cinco veces, y mostraremos por consola el valor de cada iteración, incrementándolo en cada ejecución.

```
1 var i = 1;  
2  
3 do {  
4     // mostramos el valor de i  
5     console.log(i);  
6 }
```



```
7 // incrementamos i
8 i++;
9 } while(i <= 5);
```

Al ejecutar este código en nuestra consola, veremos los números del uno al cinco, dado que en todas esas instancias la condición se cumple.

1	javascript.js:21
2	javascript.js:21
3	javascript.js:21
4	javascript.js:21
5	javascript.js:21
>	

Si ahora modificamos a propósito la condición para que sea falsa, notaremos que el bucle se cumple al menos una vez.

```
1 var i = 1;
2
3 do {
4     // mostramos el valor de i
5     console.log(i);
6
7     // incrementamos i
8     i++;
9 } while(false);
```

Como podemos ver en el resultado de la consola, siempre que la condición sea falsa, el ciclo solo se ejecutará una vez, ya que no la verifica hasta el final de la primera iteración.

1	javascript.js:21
>	



En síntesis, este ciclo nos permite controlar el flujo de nuestro código en circunstancias particulares, donde podemos ejecutar una instrucción al menos una vez, antes de considerar algunas condiciones que puedan modificar ese resultado. Al usar éste, y los demás ciclos que hemos aprendido, podremos hábilmente crear páginas web que permiten interacciones dinámicas con sus usuarios.