# Estructuras condicionales y repetitivas

* Tipo: lectura
* Formato: self-learning
* Duración: 30min

## Objetivos

* Entender qué es el control de flujo de un programa
* Entender qué son estructuras condicionales y cómo afectan el flujo del programa
* Entender qué son estructuras repetitivas y cómo afectan el flujo del programa

El texto a continuación se basa en gran medida, con ciertos ajustes, en el capítulo 2 de [Eloquent JavaScript](http://eloquentjavascript.net/),de Marijn Haverbeke, 2014. Traducción en [Español](http://hectorip.github.io/Eloquent-JavaScript-ES-online/chapters/01_values.html) disponible gracias a [hectorip](http://hectorip.github.io/), y del capítulo 6 de [JavaScript for kids](http://pepa.holla.cz/wp-content/uploads/2015/11/JavaScript-for-Kids.pdf), Nick Morgan, 2015;

## Control de Flujo

La razón por la cual creamos programas de computación es para **resolver problemas**. Un programa, hemos visto, es simplemente un conjunto de "instrucciones" que las computadoras siguen. Ese conjunto de instrucciones resuelven un problema. Hasta ahora hemos aprendido a obtener, representar, manipular y almacenar data en variables. Con esto, hemos podido crear programas que resuleven problemas muy sencillos. Esto es el caso, por ejempo, del programa que hicimos para transformar la temperatura en Celsius (°C) a Farenheit (°F). Sin embargo, para crear programas más complejos, es necesario expandir nuestro conocimiento sobre el control de flujo de un programa y las estructuras que existen para modificar este flujo.

Para utilizar la terminología correcta de JavaScript, de ahora en adelante, sustituiremos la palabra "instrucciones" por la palabra "sentencias". Una sentencia JavaScript es el equivalente a una oración en un leguaje humano. Entendemos, por lo tanto, que un programa en JavaScript es una lista de "sentencias JavaScript"; es decir, un conjunto de oraciones que le dicen al computador qué hacer.

### 1. Flujo en línea recta

Cuando un programa contiene más de una sentencia, éstas son ejecutadas de arriba hacia abajo, una por una. Por ejemplo, el siguiente programa tiene tres sentencias: la primera le declara una variable llamada name con el valor del string 'Michelle'; la segunda, imprime en la consola un mensaje personalizado de saludo 'Hola Michelle'; y la tercera, imprime en la consola un mensaje que dice 'Qué nombre tan largo tienes!'.

var name = 'Michelle';

console.log('Hola ' + name);

console.log('Qué nombre tan largo tienes!');

// returns > Hola Michelle

// Qué nombre tan largo tienes!

Una representación trivial esquemática de un flujo de control recto sería:

### 2. Flujo condicional

Ejecutar sentencias en línea recta no es la única opción que tenemos. Una alternativa es la ejecución condicional, en donde escogemos entre dos rutas diferentes basados en un valor Boolean, así:

La ejecución condicional se escribe con el keyword if en JavaScript. La instrucción if es la más simple de las estructuras de control de JavaScript. Se utiliza para ejecutar código si y sólo si una condición es verdadera. Un condicional dice: "Si algo es verdad, haz esto". Por ejemplo, si hiciste tu tarea, recibes un helado, pero si no hiciste tu tarea, no recibes el helado.

Como muestra el ejemplo a continuación, una sentencia if tiene 2 partes principales: la condición y el cuerpo. La condición debe ser un valor Boolean que va entre paréntesis. El cuerpo consiste en una o más sentencias JavaScript que se ejecutarán si, y solo si, la condición es verdadera (Boolean igual a true).

if (condición) {

// Conjunto de sentencias a ejecutar

}

Cuando necesitamos ejecutar múltiples sentencias, podemos encerrarlas en llaves ({ y }). Las llaves agrupan las sentencias, haciéndolos valer por una sola. Una secuencia de sentencias encerradas en llaves es llamada un **bloque** (de código). Muchos programadores de JavaScript encierran cada cuerpo de un if (y en los bucles, como verás más adelante), en llaves. Lo hacen en nombre de la consistencia y para evitar tener que añadir o quitar las llaves cuando el número de sentencias en el cuerpo cambie. Otros, valoran la brevedad y en el caso del if no utilizan las llaves. En este curso, siempre usaremos las llaves para ayudarnos a organizar nuestro código.

Volvamos al ejemplo anterior y agreguemos una condición antes de la sentencia que imprime el segundo mensaje:

var name = 'Michelle';

console.log('Hola ' + name);

if (name.length > 7) {

console.log('Qué nombre tan largo tienes!');

}

// returns > Hola Michelle

// Qué nombre tan largo tienes!

Con esta modificación, antes de imprimir en la consola el segundo mensaje ('Qué nombre tan largo tienes!'), el programa verifica si la longitud del string name es mayor a 7. De ser así, se imprime el segundo mensaje. De no ser así, no se ejecuta esa sentencia. En este caso, dado que Michelle tiene 8 caracteres, la condición es true. Por lo tanto, se imprime el segundo mensaje.

Cambiemos un poco este ejemplo, modificando el valor de name a 'Ana'.

var name = 'Ana';

console.log('Hola ' + name);

if (name.length > 7) {

console.log('Qué nombre tan largo tienes!');

}

// returns > Hola Ana

En esta ocasión, la condición **no** es true (es false) porque la longitud del nombre es 3, lo cual **no** es mayor a 7. Por lo tanto, el cuerpo del if **no** se ejecuta. De tal manera, únicamente se ejecuta el primer mensaje 'Hola Ana'.

#### Sentencia *if... else*

A menudo no sólo tendrás código que se ejecute cuando una condición sea verdadera, sino también que maneje el otro caso. Este camino alternativo es representado por la segunda flecha en el diagrama de flujo. La palabra clave else puede ser usada, junto con if, para crear dos rutas de ejecución separadas y alternativas.

Agregemos una sentencia else a nuestro ejemplo:

var name = 'Ana';

console.log('Hola ' + name);

if (name.length > 7) {

console.log('Qué nombre tan largo tienes!');

} else {

console.log('Tu nombre no es muy largo');

}

// returns > Hola Ana

// Tu nombre no es muy largo

Como ves, el resultado de este caso es similar al anterior, solo que si name **no** tiene una logitud mayor a 7, existe un "mensaje alternativo" que se ejecuta.

Como muestra el ejemplo a continuación, las sentencias if ... else son similares a las sentencias if, pero incluyen 2 cuerpos. Si la condición es true, se ejecutan las sentencias del primer cuerpo; de lo contrario, se ejecutan las sentencias del segundo cuerpo.

if (condición) {

Sentencias a ejecutar si condición es VERDADERA

} else {

Sentencias a ejecutar si condición es FALSA

}

Si tenemos más de dos caminos a escoger, varios pares de if...else pueden ser "encadenados". Aquí hay un ejemplo:

var num = parseInt(prompt('Dame un número', '0'));

if (num < 10){

alert('Diste un número Pequeño');

}

else if (num < 100){

alert('Diste un número Mediano');

}

else {

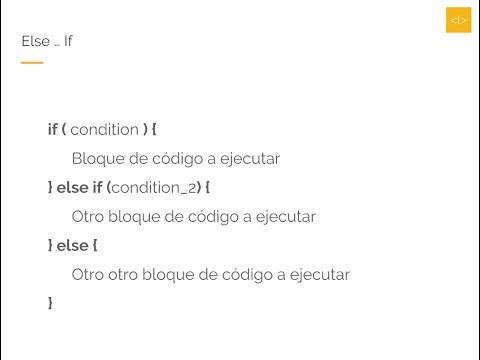
alert('Diste un número Grande');

}

El programa primero revisará si num es menor que 10. Si lo es, escoge ese camino, muestra 'Diste un número Pequeño' en un alert box y termina. Si no lo es, toma el camino del else, que en sí mismo contiene un segundo if. Si la segunda condición (< 100) se cumple, significa que el número está entre 10 y 100, y se muestra 'Diste un número Mediano' en un alert box. Si no lo es, el segundo y último elsees escogido, mostrando 'Diste un número Grande'.

El diagrama de flujo para este programa es algo así:

Veamos otro ejemplo de la aplicación de la sentencia if...else. Esta vez, con un video de otra profesora estrella de Laboratoria, Alexandra :)

[](https://www.youtube.com/watch?v=-rNwUIEQJnc)

#### Utilizando Switch

Es común ver código así:

if (variable == 'valor1') {

accion1();

}

else if (variable == 'valor2') {

accion2();

}

else if (variable == 'valor3') {

accion3();

}

else {

accionDefault();

}

Existe una estructura llamada switch que está hecha para "despachar" de un modo más directo. Desafortunadamente, la sintaxis que JavaScript usa para esto (que es heredada de la línea de lenguajes de programación de C/Java) es un poco incómoda; una cadena de sentencias if a menudo luce mejor. Aquí hay un ejemplo:

switch (prompt('¿Cómo está el clima?')) {

case 'lluvioso':

console.log('Recuerda llevar un paraguas.');

break;

case 'soleado':

console.log('Viste ligero.');

case 'nublado':

console.log('Sal a la calle.');

break;

default:

console.log('Tipo de Clima desconocido.');

break;

}

Puedes poner cualquier cantidad de etiquetas case dentro del bloque switch. El programa saltará a la etiqueta que corresponda al valor que se le dio al switch o al default si no se encuentra valor que corresponda. Se empiezan a ejecutar las sentencias desde ahí, incluso si están bajo otra etiqueta, hasta que se llegue a una sentencia break (que en español significa "detener").

En algunos casos, como en el caso de 'soleado' en el ejemplo, esto puede ser usado para compartir código entre casos (recomienda salir a la calle tanto para clima soleado como para nublado). Pero cuidado: es fácil olvidar el break, lo cuál causará que el programa ejecute código que no quieres que se ejecute.

A continuación Michelle te ayuda a entender este caso un poco mejor: [](https://www.youtube.com/watch?v=Aa0JhU6KZXs)

### 3. Flujo repetitivo: Bucles

Otra forma de modificar el flujo de un programa es estructuras repetitivas, también llamadas bucles ("loops" en inglés). Tal como hemos visto, los condicionales permiten ejecutar un conjunto de sentencias una vez si una condición ***es*** verdadera. Los bucles, permiten ejecutar un código varias veces, dependiendo de si una condición ***sigue siendo*** verdadera.

Piensa en un programa que imprima todos los números pares del 1 al 12. Una manera de escribirlo sería como sigue:

console.log(0);

console.log(2);

console.log(4);

console.log(6);

console.log(8);

console.log(10);

console.log(12);

// → 0

// → 2

// … etcetera

Eso funciona, pero la idea de escribir un programa es trabajar menos, no más. Si necesitamos todos los números menores que 1,000, lo anterior sería imposible de trabajar. Lo que necesitamos es una forma de repetir algo de código. Esta forma de **control de flujo** es llamada bucle. El control de flujo por bucles nos permite regresar a cierto punto en el programa en el que estuvimos antes y repetirlo con nuestro estado actual, tal como lo describe el siguiente diagrama:

#### Bucles while

El bucle más sencillo es el bucle while (que significa "mientras" en español). Un bucle while ejecuta repetidamente una serie de sentencias hasta que una condición particular deja de ser verdadera. Al escribir un bucle while, estás diciendo: "Sigue haciendo esto mientras esta condición sea verdadera. Detente cuando la condición se vuelva falsa."

Como muestra el ejemplo a continuación, una sentencia que comienza con el keyword while crea un bucle. Después de while viene una expresión en paréntesis que representa la condición y después un conjunto de sentencias (muy parecido al caso del if). Sin embargo, a diferencia del if, el bucle ejecuta la sentencia mientras la condición produzca un valor que sea true. Por eso es muy importante que el conjunto de sentencias incluya algo que "cambia" para que la condición eventualmente sea falsa. De lo contrario, terminarás con un ciclo infinito que nunca se detendrá, y eso usualmente es malo.

while (condición) {

Conjunto de sentencias, donde

se incluye algo que "cambia" para

que la condición eventualmente sea FALSA

}

Regresemos al problema de imprimir todos los números pares del 1 al 12, y creemos un programa que utilice un bucle while:

var number = 0;

while (number <= 12) {

console.log(number);

number = number + 2;

}

// → 0

// → 2

// … etcetera hasta 12

En este caso, como queremos imprimir los números pares del 1 al 12, creamos una variable number, la inicializamos con el valor 0, y la utilizamos en la condición. Queremos que el bloque dentro del whilecorra mientras el valor de number sea igual o menor a 12. El cuerpo del while incluye dos sentencias: la primera, imprime el número (con console.log), y la segunda incrementa number en 2 (porque queremos imprimir los pares únicamente). La variable number demuestra la forma en que una variable puede dar seguimiento al progreso de un programa. Cada vez que el bucle se repite, number se incrementa en 2. Entonces, al principio de cada repetición, el valor de la variable number es comparado con el número 12 para decidir si el programa ha hecho todo el trabajo que tenía que hacer. Es importante que entendamos que si no modificamos el valor de number con la segunda sentencia, la condición (number <=12) siempre será true y tendremos un ciclo infinito que nunca se detendrá. Esto puede causar que tu computadora tenga problemas y se congele.

Si creas un bucle infinito en uno de los ejemplos, usualmente se te preguntará si quieres detener el script después de unos cuantos segundos. Si eso falla, tendrás que cerrar la pestaña en la que estás trabajando, o, en otros nevagadores, cerrar el navegador entero para recuperarte.

Veamos algunas variantes de este mismo problema para asegurarnos que entendemos bien cómo funciona el ciclo while:

* ¿Cómo hacemos para imprimir los números **pares** del 1 al 100?: Modificamos la condición para que el bucle corra mientras number sea <= 100.

var number = 0;

while (number <= 100) {

console.log(number);

number = number + 2;

}

// → 0

// → 2

// … etcetera hasta 100

* ¿Cómo hacemos para imprimir los números **impares** del 1 al 12?: Inicializamos el valor de numbercon el valor de 1 (en lugar de 0).

var number = 1;

while (number <= 12) {

console.log(number);

number = number + 2;

}

// → 1

// → 3

// … etcetera hasta 11

* ¿Cómo hacemos para imprimir todos los números **pares** e **impares** del 1 al 12?: En la segunda sentencia del while, incrementamos number en 1 (en lugar de 2) cada ciclo.

var number = 0;

while (number <= 12) {

console.log(number);

number = number + 1;

}

// → 0

// → 1

// → 2

// … etcetera hasta 12

#### Bucles do

El bucle do es una estructura de control similar al bucle while. Se diferencia en sólo un punto: un bucle do siempre ejecuta su cuerpo **por lo menos una vez** y empieza a verificar si debería parar sólo después de la primera ejecución. Para reflejar esto, la condición aparece después del cuerpo del bucle. Veamos un ejemplo:

do {

var yourName = prompt('¿Quién eres?');

} while (!yourName);

console.log('tu nombre es ' + yourName);

Este programa te obligará a introducir un nombre. Preguntará una y otra vez hasta que obtenga algo que no sea una cadena vacía. Aplicar el operador ! convierte un valor a Boolean negándolo y todas las cadenas excepto '' se convierten a true. Esto significa que el bucle continúa corriendo hasta que des un nombre que no sea una cadena vacía.

#### Bucles *for*

Muchos bucles siguen el patrón de los ejemplos previos del while. Primero, una variable “contador” es creada para dar seguimiento al progreso del bucle. Luego viene el bucle while, cuya expresión condicional normalmente verifica si el contador ha alcanzado cierto límite. El final del cuerpo del bucle, el contador es actualizado para dar seguimiento al progreso.

Debido a que este patrón es tan común, JavaScript y otros lenguajes similares proveen una versión un poco más corta y más completa: el bucle for.

for (var number = 0; number <= 12; number = number + 2){

console.log(number);

}

// → 0

// → 2

// … etc.

Este programa es exactamente equivalente al ejemplo previo de impresión de números pares. El único cambio es que todas las sentencias que están relacionadas con el "estado" del bucle están agrupadas.

Los paréntesis después del keyword for tienen que contener dos puntos y coma (;). La parte que está antes del primer punto y coma inicializa el bucle, normalmente al definir una variable. La segunda parte es la expresión que verifica si el bucle tiene que continuar. La parte final actualiza el estado del bucle antes de cada iteración. En la mayoría de los casos, esto es más corto y claro que una construcción con while.

Aquí está un código que calcula 2^10 (2 exponencial 10), usando el bucle for:

var result = 1;

for (var counter = 0; counter < 10; counter = counter + 1){

result = result \* 2

}

console.log(result);

// → 1024

La representación general del bucle for es la siguiente:

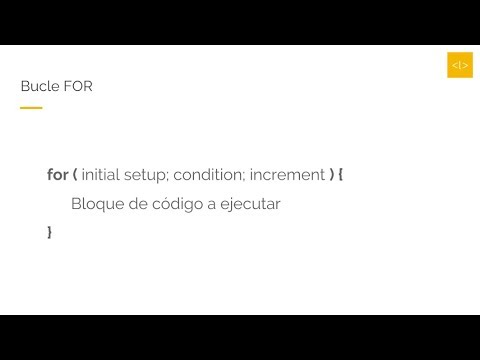
for (initial setup; condition; increment){

Bloque de código a ejecutar

}

El initial setup (por ejemplo: var counter = 0) se ejecuta antes de que se inicie el bucle. Generalmente se usa para crear una variable que rastrea el número de veces que se ha ejecutado el bucle. La condition(counter < 10) se comprueba antes de cada ejecución del cuerpo de bucle. Si la condición es verdadera, el cuerpo es ejecutado; si es falsa, el bucle se detiene. En este caso, el bucle se detendrá una vez que counter ya no sea inferior a 10. El increment (counter = counter + 1) se ejecuta después de cada ejecución del cuerpo de bucle. Generalmente se utiliza para actualizar la variable de bucle. En nuestro ejemplo, lo usamos para agregar 1 a counter cada vez que se ejecuta el bucle.

A continuación Alexandra te ayuda a entender el flujo de un bucle for que imprime el texto 'Hola mundo!' varias veces:

[](https://www.youtube.com/watch?v=lKwx1RAxTfo)

#### Forzando la salida de un bucle

Hacer que la condición del bucle produzca false no es la única forma de que un bucle termine. Podemos usar la sentencia especial break, utilizada en switch, que tiene el efecto de salir inmediatamente del bucle que la esté encerrando.

El siguiente programa ilustra el uso de la sentencia break para salir de un bucle. Queremos hacer un programa que encuentre el primer número que es más grande o igual que 20 y divisible por 7.

for (var current = 20; ; current++) {

if (current % 7 == 0)

break;

}

console.log(current);

// → 21

Usar el operador de sobrante o módulo (%) es una forma fácil de probar si un número es divisible por otro. Si lo es, el sobrante de la división es cero. También [recuerda](https://github.com/Laboratoria/curricula-js/blob/master/01-intro/02-variables-and-data-types/02-variables.md) que current++ es lo mismo que current = current + 1.

El for en este ejemplo no tiene la parte que verifica si el bucle debe terminar. Esto significa que el loop nunca terminará hasta que la sentencia break que está adentro sea ejecutada.

Como, hemos visto, si dejaras afuera esa sentencia break o accidentalmente escribieras una condición que siempre produzca true, tu programa se quedaría atorado en un bucle infinito.

La palabra clave continue es similar a break en que influencia el progreso del bucle. Cuando se encuentra continue en el cuerpo de un bucle, el control sale del curpo del bucle inmediatamente y continúa en la próxima iteración del bucle.

A continuación Michelle te ayuda explica, con un ejemplo, las aplicaciones de break y continue dentro de un for: