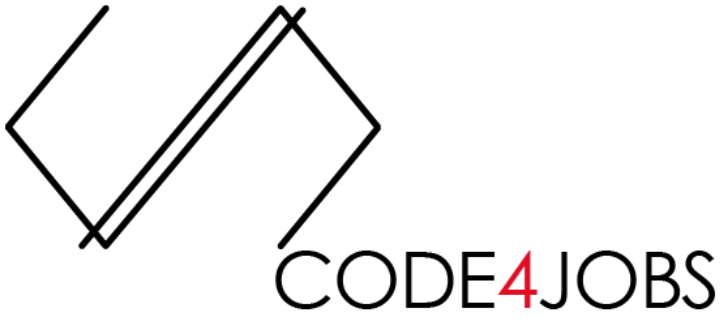


bbk bootcamp

Curso previo de JavaScript

Parte 3

EL OBJETIVO

El objetivo de este segundo módulo de ejercicios es aprender a crear programas que no ejecuten siempre las mismas instrucciones sino que puedan mostrar resultados distintos cuando introducimos datos diferentes. Conseguir que el programa tome decisiones diferentes dependiendo de varios factores hace que podamos controlar el flujo del programa y creemos programas más versátiles y útiles. Por supuesto, la práctica que esto supone hará que continúes familiarizándote con la herramienta de Visual Studio Code y a seguir aprendiendo las bases del principal lenguaje de programación que utilizaremos durante el Bootcamp, JavaScript.

Continuaremos añadiendo distintas partes del léxico de JavaScript a nuestro vocabulario a través de ejemplos y ejercicios. A la vez que lees este documento, consulta los programas que están explicando y no tengas miedo de experimentar introduciendo distintos cambios en los ejemplos para ver que cambia. Al mismo tiempo, ten en cuenta que estamos todavía aprendiendo los aspectos más básicos de JavaScript, por lo que no te preocupes si ves que no eres capaz de traducir ideas que tienes para programas complejos en programas, eso llegará cuando estemos trabajando en el propio Bootcamp.

Por supuesto y como siempre, si en algún momento tienes cualquier tipo de problema, no dudes en ponerte en contacto con nosotros.

Para cualquier duda que tengas, puedes:

E-mail

Escribir a cualquiera de los Instructores con tus dudas:

- peio@code4jobs.com
- erlantz@code4jobs.com

Discord

Conectarte al Discord de Code4Jobs

- Compartir tu duda en el canal #2022-Diciembre
- Escribir por privado a cualquiera de los instructores en el Discord.

Proyectos necesarios

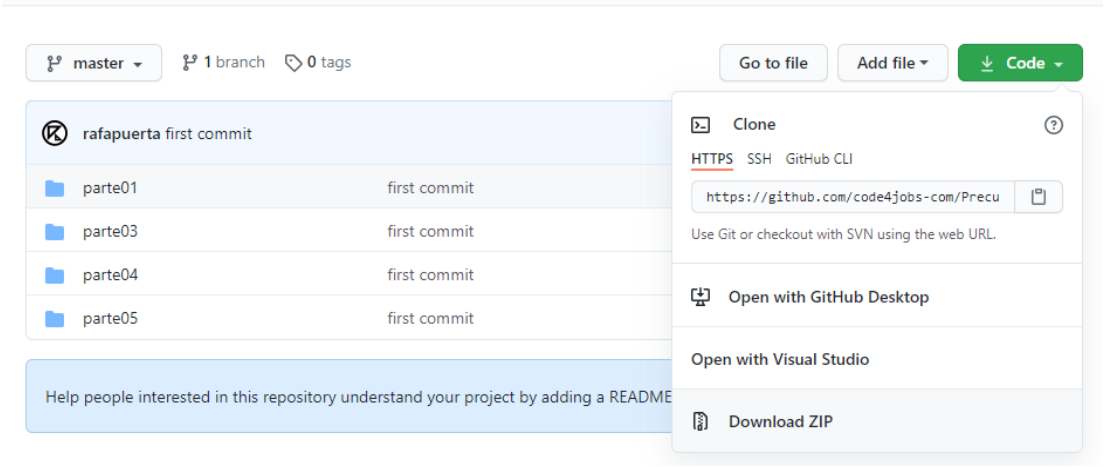
En primer lugar, tendremos que cargar en el Visual Studio Code los proyectos relacionados con los siguientes ejemplos y ejercicios.

En la parte 1 de Precurso ya descargamos todos los ejercicios necesarios y ahora usaremos los que están contenidos en la carpeta **parte03**

Si en algún caso no dispones de los archivos, no te preocupes, puedes volver a bajarlos siguiendo estos pasos:

1- Haz clic en este enlace: https://github.com/code4jobs-com/Precurso-Presencial-JS

2- Haz clic en el botón verde **Code** y después en la opción **Download Zip**



3- Abre la carpeta en el Buscador o el Explorador de Archivos y en el caso de Windows haz clic derecho y elige “Abrir con Visual Studio Code” o en el caso de MacOs arrastra la carpeta al icono de Visual Studio Code para que se abra con ella cargada.

VARIABLES BOOLEANAS

A lo largo de los ejercicios vistos en la sección Introducción a JavaScript vimos dos tipos de valores que podemos guardar en variables:

- Valores de cadenas de textos `"string"`
- Valores de números `number`

Ahora veremos un tipo de valor mucho más limitado. El tipo booleano (`boolean`) solo puede tener dos valores, **true** o **false** (verdadero o falso). Por lo tanto, podemos utilizar las variables tipo bool para llevar un control de las opciones que solo pueden tener dos valores.

Como con todas las variables, la declaramos con la palabra clave *let* y le asignamos un valor booleano para que se guarde en la variable. En el caso de las variables booleanas, este valor tendrá que ser la palabra clave *true* o *false*, que asignamos con el operador de asignación (=):

```
let test;
test = false;
```

Como con las otras variables, podemos mostrarlo en pantalla utilizando console.log y podemos concatenarlo a cadenas de caracteres utilizando el operador de concatenación (+):

```
console.log(test);
let mensaje;
mensaje = "La variable test contiene el valor " + test;
console.log(mensaje);
```

01

Podéis ver el ejemplo de usarlo en la carpeta **01_Ejemplo_Bool** . Es importante escribir ***true*** y ***false*** exactamente así, sin mayúsculas, ya que al ser palabras clave tienen que escribirse de ese modo o JavaScript no entenderá lo que son y el valor será incorrecto.

Para obtener valores booleanos (y que no tengamos nosotros que asignar true o false siempre que usamos una variable bool), podemos utilizar operadores de igualdad y relacionales. Estos operadores comparan las dos variables y dan como resultado true o false dependiendo del operador y el valor de las variables.

👉Recuerda que para esta parte del precurso usaremos los archivos dentro de la carpeta **parte03**

Operadores relacionales

- `<` valor izquierdo menor que el derecho
- `<=` valor izquierdo menor o igual que el derecho
- `>` valor izquierdo mayor que el derecho
- `>=` valor izquierdo mayor o igual que el derecho
- `==` valor izquierdo igual que el derecho
- `!=` valor izquierdo no igual al derecho

Ya que utilizamos el operador **=** para asignar valor, debemos utilizar el operador **==** para ver si dos valores son iguales.

Veamos algunos ejemplos:

- `3 < 5` , 3 es menor que 5, por lo tanto `true`
- `5 < 3` , 5 no es menor que 3, por lo tanto `false`
- `4 < 4`, 4 no es menor que 4 ,por lo tanto `false`
- `4 <= 4` , 4 es igual a 4, por lo tanto 4 es menor o igual a 4, y por lo tanto `true`
- `4 <= 5` , 4 es menor que 5, por lo tanto 4 es menor o igual que 5, por lo tanto `true`
- `4 >= 4` , 4 es igual que 4, por lo tanto 4 es mayor o igual que 4, por lo tanto `true`
- `5 >= 4` , 5 es mayor que 4, por lo tanto 5 es mayor o igual que 4, por lo tanto `true`
- `4 == 3` false , 4 no es igual a 3, por lo tanto `false`
- `4 == 4` true , 4 es igual a 4, por lo tanto `true`
- `4 != 3` true , 4 es desigual a 3, por lo tanto `true`
- `4 != 4` false , 4 no es desigual a 4, por lo tanto `false`

Mostrar el resultado de estas operaciones en consola tiene una dificultad. Podemos intentar utilizar uno de estos operadores dentro de console.log de este modo:

```
console.log("3 > 5 es " + 3 > 5);
```

El problema es que el resultado de este console.log sería `false`. Lo que queremos que haga la instrucción es comparar si 3 es mayor que 5, y concatenar el resultado a la cadena de caracteres "3 > 5 es " . Pero lo que JavaScript está haciendo primero en cambio es unir la cadena "3 > 5 es " a 3. Por lo tanto, en el siguiente paso intenta ver si "3 > 5 es 3" es mayor que 5 por lo que dice que es falso. Ya que queremos alterar el orden de operaciones, utilizaremos las paréntesis. Por lo tanto la manera correcta de mostrar en pantalla lo que queríamos es:

```
console.log("3 > 5 es " + (3 > 5));
```

02

En el .html dentro de la carpeta **02_Ejemplo_Relacionales** puedes ver una serie de comparaciones mostradas en consola.

03

Veamos ahora un programa en el que utilizamos las variables booleanas. Vamos a preguntarle al usuario su edad. Después Mostraremos en pantalla el mensaje `"Eres mayor de edad: True"` si tiene 18 años o más o `"Eres mayor de edad: False"` en caso contrario- Puedes ver el programa completo en **03_Ejemplo_Relacionales_Input** , veamos está creado.

Primero, vamos a mostrar el mensaje:

```
prompt("¿Cuantos años tienes?");
```

También crearemos una variable de nombre edad, en el que guardaremos la edad que teclee el usuario.

```
let edad;
```

Ya que tenemos que obtener lo que teclee el usuario, sabemos que queremos utilizar console.log() , pero recordemos que prompt captura valores de tipo string, así que para convertirlo en un valor de número entero tenemos que usar parseInt() . Por lo tanto, vamos a utilizar esta instrucción:

```
edad = parseInt(prompt("¿Cuantos años tienes?"));
```

Por lo tanto, parseInt convierte lo capturado del teclado por prompt, y pone ese valor dentro de la variable edad. Podríamos haberlo hecho de esta manera también:

```
let respuesta = prompt("¿Cuantos años tienes? ");
```

```
edad = parseInt(respuesta);
```

Estas dos instrucciones hacen exactamente lo mismo que

```
edad = parseInt(prompt("¿Cuantos años tienes?"));
```

Finalmente, llegamos al objetivo. Vamos a querer ver si edad es mayor o igual que 18 para decir que

es mayor de edad. Para ello utilizaremos `edad >= 18` . Podríamos mostrarlo en pantalla de este modo:

```
console.log("Eres mayor de edad: " + (edad >= 18));
```

Una vez más, podemos hacer lo mismo de otro modo:

```
let mayorDeEdad;
mayorDeEdad = edad >= 18;
console.log("Eres mayor de edad: " + mayorDeEdad);
```

Recordar que durante el pre-curso estamos viendo algunas cosas de manera más simple. Durante el Bootcamp podréis crear código de modo que si el usuario decide introducir texto en vez de un número, el programa no guardará `NaN`. Pero de momento, tenemos que tener cuidado de meter sólo valores numéricos cuando lo necesitemos.

CONDICIONALES

Todos los programas que hemos creado hasta ahora empiezan en la primera instrucción y ejecutan todas y cada una de las instrucciones hasta llegar a la última. Ahora veremos un modo de poder no ejecutar ciertas instrucciones, si algunas condiciones no se cumplen.

Utilizaremos la palabra clave ***if*** para indicar a JavaScript que solo tiene que ejecutar ciertas instrucciones si una condición se cumple. Esta condición es un valor booleano que ponemos entre paréntesis después de if. `if(true)` por ejemplo siempre ejecutaría las instrucciones que le siguen entre las llaves `{}`

```
if(true) {
  console.log("Siempre se ejecuta esto");
}
```

En cambio con false, nunca se ejecutaría:

```
if(false) {
  console.log("Nunca se ejecutará");
}
```

En este caso, VS Code no lo marcará como un error, pero sí avisará de que es código que nunca se va a ejecutar. Obviamente, los dos ejemplos anteriores no son útiles. En el caso de código que siempre queramos ejecutar podemos simplemente escribir la instrucción de modo normal. En el caso de código que nunca queramos ejecutar no tendríamos porqué escribir nada.

Pero si sabemos que como condición de if debemos poner un valor de true o false, sabemos que podríamos poner en su lugar una operación de relación o de igualdad, o una variable booleana. Por ejemplo:

```
if(3 < 5) {
  console.log("3 es menor que 5.");
}
```

Sabemos que el resultado de una operación con `<` siempre será de true o false. Por lo tanto, podemos utilizar una expresión de este tipo (o con `<=`, `>`, `>=`, `==` o `!=`) y las instrucciones que siguen al if se ejecutarán si el resultado de la operación es verdadero. Este ejemplo concreto, una vez más, no es útil. Obviamente, `3 < 5` siempre tendrá como resultado true, por lo que siempre se ejecutará el código. Veamos un ejemplo útil:

```
if(edad >= 18) {console.log("Eres mayor de edad.");}
```

En este caso, la instrucción se ejecutará dependiendo del valor de la variable edad. Existe la posibilidad de que le hayamos asignado un valor que no cambia durante todo el programa, en cuyo

caso volvemos a tener una instrucción que se ejecutará o siempre o nunca.

Pero si obtenemos el valor de edad cuando el usuario lo teclea, podemos ver que `console.log("Eres mayor de edad.")` se ejecutará cuando el usuario diga que la edad es igual o mayor que 18, y no se ejecutará en el caso contrario.

En los ejemplos de if vistos ahora ahora, sólo hemos puesto una instrucción entre { y }, pero podríamos tener tantas como quisiéramos, y en el caso de que el valor en el if fuera false, el programa ignorará todos.

También podemos utilizar una variable booleana como condición en el if. Pongamos por ejemplo que tenemos una variable bool vip; que nos indica si alguien es VIP, true en caso de que lo sea, false en el caso contrario. Podemos pensar en comprobarlo de este modo:

```
if(vip == true) {
  console.log("Cliente VIP.");
}
```

Estamos comprobando que el valor dentro de la variable sea true. Si es false, la comparación de valores entre true y false da resultado de que no es igual y por lo tanto el resultado es false y la instrucción no se ejecuta. Aunque podamos hacer esto y funcione correctamente, funcionará igualmente si lo hiciéramos así:

```
if(vip) {
  console.log("Cliente VIP.");
}
```

Si vip contiene un valor true, se ejecutará y en caso contrario no. Si quisiéramos hacer la comprobación contraria, es decir, comprobar si vip contiene el valor false, utilizamos ! para ver el valor contrario de la variable booleana, haciendo así que la condición if sea verdadero si la variable tiene valor falso:

```
if(!vip) {
  console.log("Cliente no VIP");
}
```

En el último ejemplo, hemos visto cómo podríamos ver si alguien es VIP o no:

```
if(vip) {
  console.log("Cliente VIP."); }if(!vip) {
  console.log("Cliente no VIP.");
}
```

Veamos exactamente lo que está haciendo el código. En primer lugar mirar el valor de vip y si es true , muestra en pantalla el mensaje "Cliente VIP.". Después mira el valor de vip y si es false, muestra en pantalla el mensaje "Cliente no VIP.". Aunque el código hace lo que queremos (mostrar un mensaje u otro dependiendo del valor de vip), preferimos si el programa mirara el valor de vip una vez y muestra un mensaje si el valor es true y otro si no lo es.

JavaScript nos permite crear código para hacer esto con el formato if ... else ... Veamos cómo funciona:

```
if(vip) {
  console.log("Cliente VIP.");
}else {
  console.log("Cliente no VIP.");
}
```

Esta vez, sólo comprobamos el valor de la variable vip una vez. Si es true, se ejecuta la instrucción de mostrar a pantalla "Cliente VIP." e ignoramos la parte de else. Si es false, pasamos directamente a la parte de else, y mostramos a pantalla "Cliente no VIP.".

06

Ejercicio mayor de edad (carpeta **06_Ejercicio_If_Else)**

En el archivo html dentro de la carpeta **06_Ejercicio_If_Else**, dónde pone que introduzcas tu código, deberás hacer que el programa le pregunte al usuario su edad. Si es igual o mayor a 18, mostrarás en consola "Eres mayor de edad" y en caso contrario "Eres menor".

Imaginemos ahora que queremos hacer un programa en el que le decimos al usuario que tiene tres intentos para adivinar un número entre el 1 y el 10. La respuesta correcta (el número que tiene que adivinar) es el 5. Por lo tanto creamos una variable y le damos un valor de 5.

```
let correcta; correcta = 5;
```

También necesitamos una variable en la que guardar la respuesta del usuario, por lo que creamos otra variable:

```
let respuesta;
```

Después le mostramos al usuario en pantalla el mensaje Adivina el número entre 1 y 10 y cogemos la respuesta, la transformamos en número entero (la parseamos con parseInt()) y asignamos ese valor a la variable respuesta:

```
respuesta = parseInt(prompt("Adivina el número entre 1 y 10"));
```

Ahora si la respuesta es igual que la respuesta correcta, queremos mostrar en pantalla que ha acertado el número. Si no es así, queremos volver a pedir otro número. Por lo tanto la estructura será:

- Pedir un número y guardarlo en una variable

- Si el número es el correcto se muestra en pantalla que acertó - Si no es correcto pedimos otro número y lo guardamos

- Si el número es el correcto se muestra en pantalla que acertó - Si no es correcto pedimos otro número y lo guardamos

- …

07

En el proyecto **07_Ejemplo_If_Else_Anidado** puedes ver este ejemplo concreto. Fijate cómo dentro de cada else estamos repitiendo código. Ahora imagina que decimos que en vez de tres intentos vamos a dejarle al usuario diez intentos. ¿Ves que tendríamos que repetir muchísimo código? Ahora imagina que una vez has cambiado el código, decidimos que en vez de adivinar un número entre el 1 y el 10, tienen que adivinar entre el 1 y el 20. ¿Ves cómo tendríamos que cambiar cada vez que hemos escrito ese mensaje en pantalla? Y si nos despistamos y nos olvidamos de cambiar una vez en la que sale, tendríamos un error en el código.

Una mejor manera de hacer algo repetitivo es hacer que sea precisamente el ordenador el que repita el código. Veremos un modo más de usar if y después veremos cómo podemos hacerlo.

Supongamos ahora que estamos creando un menú en nuestro programa. Las opciones serán *insertar dato*, *editar dato*, *eliminar dato* y *salir*, y cada uno de ellos tendrá un número asociado. El usuario introduce su opción y comparamos el valor que llega con las distintas opciones:

```
if(opcion == 1) {
...
}
if(opcion == 2) {
...
}
if(opcion == 3) {
...
}
if(opcion == 4) {
...
}
```

Ya sabemos que este no es el mejor método, ya que si por ejemplo opcion tiene el valor 1, aún así compararemos si tiene 2, 3, o 4. Por lo tanto, utilizamos un else como antes:

```
if (opcion == 1) {
  //opción 1 seleccionada
} else {
  if (opcion == 2) {
      //opción 2 seleccionada
  } else {
      if (opcion == 3) {
      //opción 3 seleccionada
      } else {
      if (opcion == 4) {
      //opción 4 seleccionada
      }
      }
  }
}
```

Pero puedes ver que cuantas más opciones haya el código será más difícil de seguir. Es por esto que hay otra manera de escribir, utilizando else if:

```
if(opcion == 1) {
// Opción 1 seleccionada
} else if(opcion == 2) {
// Opción 2 seleccionada
} else if(opcion == 3) {
// Opción 3 seleccionada
} else if(opcion == 4) {
// Opción 4 seleccionada
} else {
// Opción erronea
}
```

08

Como ves else if permite *encadenar* comparaciones de if en el mismo nivel lo que hace que el código sea más fácil de leer. Puedes ver el ejemplo en la carpeta **08_Ejemplo_Else_If**

Veamos ahora otro modo de hacer exactamente lo mismo que antes, utilizando la instrucción switch . Este es el formato que hay que seguir:

```
switch(variable)
{
case 1:
// instrucciones a ejecutar si la variable // es igual a 1
break;
case 2:
// instrucciones a ejecutar si variable es      igual a 2
break;
...
default:
// instrucciones a ejecutar si el valor de variable // no coincide con
ninguno de los valores en los case. break;
}
```

Se utiliza break para salir de la operación donde estamos. La mayor diferencia entre utilizar un switch y un if ... else if ... else es que los valores en switch tienen que coincidir exactamente, mientras que en un if puedes poner un if(x<5) y crear así rangos de valores. Tal como en el if ... else if ... else el else final aceptaba valores que no se habían visto antes, el default es donde pondremos instrucciones en caso de que ninguno de los valores anteriores se encuentre.

09

Veamos por ejemplo como se puede hacer el ejemplo del menú anterior con un switch (puedes ver el html en 09_Ejemplo_Switch para ejecutarlo).

```
switch(opcion) {
  case 1:
    console.log("Elegiste insertar dato"); break;
  case 2:
    console.log("Elegiste editar dato"); break;
  case 3:
    console.log("Elegiste eliminar dato"); break;
  case 4:
    console.log("Elegiste salir del programa"); break;
  default:
    console.log("Opción erronea"); break;
}
```

BUCLES WHILE

En todos los programas anteriores, aunque hayamos podido ignorar ciertas partes del código, cada una de las líneas se ha ejecutado como mucho una vez. Para tareas repetitivas es conveniente utilizar un bucle para que sea el mismo ordenador el que repita las instrucciones (en vez de tener que escribir código repetido varias veces).

Una de las maneras de repetir código es utilizar la instrucción **while** (*mientras* en inglés). while tiene un valor booleano entre paréntesis (del mismo modo que if), y si es true ejecuta el código que tiene dentro. La diferencia es que al llegar al final del bloque de instrucciones, *vuelve* a comparar si sigue siendo true el valor booleano. Si es así, vuelve a ejecutar las instrucciones, y las seguirá repitiendo hasta que en alguna de las ocasiones que acabe de ejecutar las instrucciones dentro y compare el valor este sea false. Veamos un ejemplo.

Digamos que tenemos un valor de número entero dentro de una variable que llamaremos x. Asignaremos 1 a x:

```
let x;
x = 1;
```

Mientras x sea menor o igual que 3, mostraremos en pantalla el valor de x y después le sumaremos uno a x:

```
while(x <= 3) {
  console.log(x);
  x = x + 1;
}
```

10

Tienes este código en la primera parte del proyecto **10_Ejemplo_While_1**.
La primera vez que el código llega al while, x tiene un valor de 1. 1 es menor o igual que 3, por lo tanto JavaScript ejecuta el console.log y ejecuta la instrucción donde x coge el valor de x (1) más 1, por lo que x tiene 2. Ya que hemos ejecutado todas las instrucciones dentro del while, volvemos a comparar si x es menor o igual que 3. Ya que tiene un valor de 2, esta operación da un resultado de true y volvemos a ejecutar el console.log y el x = x + 1. Esta vez, x tiene un valor de 3. Sigue siendo menor o igual que 3, por lo que una vez más mostramos el valor de x en pantalla y le añadimos uno a x. Esta vez, al ser x igual a 4, x **NO** es menor o igual que 3, por lo que al ser false no ejecutamos las instrucciones dentro del while y continuamos a las siguientes instrucciones del programa. Por lo tanto, el resultado de ejecutar el while es

```
1
2
3
```

Como hemos dicho anteriormente, el ordenador sigue las instrucciones que le escribas ciegamente. Por ello a veces podemos tener problemas. Por ejemplo, digamos que en el while anterior había escrito lo siguiente:

```
while(x <= 3) {
  console.log(x);
}
```

Comenzábamos con un valor de x de 1. Por lo tanto al llegar al while el programa comprobaría que x es menor o igual que 3 y pasaría a mostrar en pantalla el número 1. Después, volvería a comprobar que x es menor o igual que 3. Como x no cambia, esta condición *siempre* será true, y el programa mostrará el número 1 en pantalla sin parar. Precisamente para parar un programa así tendríamos que pararlo, utilizando el botón de parar.

Por lo tanto, tener siempre cuidado de si vais a hacer que algo se repita, que dentro del código que se repite se cambie lo que se compara dentro del while o el programa se quedará en bucle infinito y tendréis que parar el programa forzandolo (si queréis, cambiar el cambio del valor x para comprobar cómo tenéis que cerrar la ventana del navegador y que se queda sin responder un buen rato).

Otro modo de que el bucle pare (de manera normal dentro del programa) es utilizar la instrucción break. Esto hace que el bucle pare inmediatamente y el programa pare. Veamos un ejemplo (lo tienes en la segunda mitad del proyecto **10_Ejemplo_While_1**):

```
x = 1; while(x <= 3) {
  if(x == 2) {
    break;
  }
  console.log(x);
  x = x + 1;
}
```

Primero volvemos a hacer que x tenga el valor 1. Dentro del bucle, primero miramos si x es igual a 2. Como no lo es, pasamos a mostrar en pantalla el valor de x (1) y añadimos uno a x haciendo que valga 2. Volvemos al while, y como 2 es menor o igual que 3, entramos dentro de las instrucciones. Comparamos que x sea 2 y como esta vez lo es ejecutamos la instrucción break. Esto hace que salgamos del while inmediatamente (no ejecutamos el log de después, directamente salimos de las instrucciones de dentro del bucle). Por lo tanto el resultado de esto es simplemente mostrar 1 en consola.

Ahora que conocemos las instrucciones de while y break, podemos volver a escribir el programa en el que dábamos tres intentos al usuario a adivinar un número entre el 1 y el 10.

11

Puedes verlo en el **11_Ejemplo_While_2** . Aunque tengamos que crear dos variables más (**intento** para saber en qué **intento** estamos y **max_intentos** para marcar el máximo número de intentos) los utilizamos los dos en el while para saber cuando parar (es decir cuando el número de **intento** es mayor que **max_intentos** no repetimos el código) y, por supuesto, recordar que tenemos que aumentar el valor de **intento** al final de cada bucle para que el programa no se acabe repitiendo sin parar.

Puedes ver cómo dentro del bucle hacemos un if/else normal, y no tenemos que poner múltiples if/else dentro de los else para cuando no aciertan el número a adivinar.

```
while(intento <= max_intentos) {
  console.log();
  respuesta = parseInt(prompt("Adivina el número entre 1 y 10"));
  if(respuesta == correcta) {
    console.log("Has adivinado el número en el intento " + intento);
    break;
  } else {
    console.log("Número incorrecto");
  }
  intento = intento + 1;
}
```

En este ejemplo puedes ver cómo si quisiéramos cambiar el número de intentos, o el texto que se muestra en pantalla, sería cuestión de cambiarlo en un solo sitio y no habría ningún problema.

Una limitación del bucle while es que siempre tiene que comprobar la condición antes de ejecutar las instrucciones que tiene dentro. Veamos por ejemplo un programa en el que le pedimos al usuario un número. Si el número es 0, acabamos el programa, si no, mostramos el número que han introducido en pantalla.

12

Para el siguiente programa (lo puedes ver en **12_Ejemplo_Do_While**) vamos a querer repetir una serie de instrucciones mientras el valor de la variable **numero** no sea 0. Por lo tanto sabemos que tenemos

```
while(numero != 0){
  ...
  console.log(numero);
}
```

Y dentro del while tenemos las instrucciones, pero nos encontramos con un problema. Después de mostrar el número, queremos pedirle *otro* número al usuario. Eso significa que el bucle acaba así:

```
while(numero != 0) {
  numero = parseInt(prompt("Introduce un número: "));
  console.log(numero);
}
```

Pero si te fijas, cuando hemos llegado al bucle necesitamos tener un valor dentro de numero. Si pusiéramos un valor que nosotros le indicamos en el código, por ejemplo numero = 1; nos arriesgamos a que más adelante decidamos que no queremos que se repita el código cuando numero es igual a 1. Es por ello que no conviene usar números aleatorios. Por lo tanto para que el programa funcione correctamente, tenemos que repetir código:

```
numero = parseInt(prompt("Introduce un número:"));
while(numero != 0) {
  console.log(numero);
  numero = parseInt(prompt("Introduce un número:"));
}
```

Ahora el programa funciona como queremos, pero tenemos el mismo problema que siempre tenemos cuando hay código repetido: si queremos cambiar la frase "Introduce un número:" por cualquier otra, tendriamos que cambiarla en dos sitios, y es posible que acabemos cambiandola solo en una y no en las dos, dejando un error en el programa.

Para este tipo de problema, podemos utilizar la instrucción do {...} while(...); que precisamente hace que las instrucciones de dentro se ejecuten *al menos una vez* y después ya repite el código si la condición dentro del while se cumple. Veamos un ejemplo (en la segunda mitad de **12_Ejemplo_Do_while**):

```
do {
  numero = parseInt(prompt("Introduce un número:"));
  console.log(numero);
} while(numero != 0);
```

Este código hace exactamente lo mismo que el anterior, pero sin tener que repetir código. Para acordarte de la diferencia, fijate que miramos la condición cuando llegamos al while(...). En el caso del while normal, la condición aparece arriba, por lo que si no se cumple al llegar por primera vez a esa parte del código, las instrucciones dentro del bucle no se ejecutarán ni una vez. En el caso del do {} while() pudes ver que *primero* se ejecutarán las instrucciones dentro del bucle, y *después* se comprobará la condición. Por lo tanto, si tienes un bucle en el que *siempre* quieres ejecutar las instrucciones al menos una vez, utiliza el do {} while();

13

Ejercicio cuenta atrás (proyecto 13_Ejercicio_While)

En este ejercicio queremos que se muestre en pantalla una cuenta atrás de 10 a 0 (usando un while) y al llegar a 0 que muestre el texto **¡Despegue!**

BUCLES FOR

Hemos visto que while se puede utilizar tanto para repetir algo un número específico de veces como para repetir código hasta que algo cambia (como el ejemplo en el que parábamos si el usuario introducía un 0). Normalmente, si sabemos cuántas veces queremos repetir el código, utilizamos *for* ya que es más sencillo. Para ver cómo funciona for veamos cómo repetimos el código un número específico de veces con while:

```
let numero = 1;
while (numero <= 5) {
  console.log("El número es " + numero);
  numero = numero + 1;
}
```

Podríamos clasificar las partes del código anterior de este modo:

```
let numero = 1;
numero <= 5
numero = numero + 1;
```

- Estamos inicializando un contador

- La condición para que se repita el bucle - Cambiamos el contador en cada vuelta

Y son precisamente estas tres partes las que utiliza el for en este formato:

```
for(inicializar; condición; cambio) {
// código a repetir aquí
}
```

Veamos lo mismo pero utilizando la variable i en vez de la variable numero en un for:

```
for(let i = 1; i <= 5; i = i + 1) {
  console.log("El valor de i es " + i);
}
```

Aunque se escriban de modos diferentes, el efecto de utilizar el for será exactamente el mismo que con el while, excepto por una diferencia. Ambos repetirán el código 5 veces (en este caso mostrando un mensaje a pantalla). Pero el valor de la variable numero s seguirá existiendo después de que acabe el bucle, en cambio i dejara de existir.

Este hecho tiene que ver con un concepto que todavía no hemos visto, denominado *ámbito de variable*. En todos los programas que hemos visto hasta ahora las variables que hemos creado podían utilizarse a lo largo de todo el programa. Pero como hemos comentado, hay programas con cientos de miles de líneas de código en total. Si además pensamos que hay múltiples grupos de

gente trabajando en ese código, ¿cuál es la probabilidad de que dos personas intenten utilizar el mismo nombre de variable en dos lugares distintos? Podéis ver que esto podría causar todo tipo de problemas.

En JavaScript cuando creamos una variable con la palabra clave *let* esa variable existe en una parte del programa. Esta parte está delimitada por las llaves { y } que la rodean. Por supuesto, no podemos utilizarlo antes que hayamos creado la variable, pero podemos seguir utilizándolo hasta que lleguemos a la llave que cierra.

En este caso, por cómo funciona el for, la variable i puede considerarse que se crea dentro de las llaves del for, por eso al acabar no puede utilizarse. Como la variable numero se crea antes que el while, sigue existiendo después del mismo.

14

En el proyecto **14_Ejemplo_For** puedes ver un ejemplo tanto del for como el while y verás que también mostramos cómo numero sigue existiendo, pero intentar utilizar i da un resultado *undefined* que es como JavaScript marca una variable que no tiene valor / no existente.

Una de las principales ventajas del for respecto al while es cuando el código que repetimos es muy grande. Imagina que tenemos 50 instrucciones que tenemos que repetir varias veces. Si utilizaramos un while tendríamos la inicialización (let numero = 1;) justo antes del while, la condición (numero <= 5) en el propio while, pero el cambio (numero = numero + 1;) 50 líneas más abajo. Por esta razón, en general, cuando queremos repetir código un número determinado de veces utilizamos un for y cuando queremos repetir código *hasta que una condición específica no se cumpla* utilizamos un while.

15

Para repasar las posibilidades que tenemos con el bucle for, veamos distintos bucles. Para verlos, puedes abrir el archivo dentro de 15_Ejemplo_For_2 .

En el primer ejemplo empezaremos iniciando el valor de la variable i a 1, seguiremos realizando el bucle mientras i sea menor que 10 y en cada vuelta de bucle le añadiremos 1 a i. Dentro del bucle solamente mostramos el valor de i en la pantalla. Si vas probando el valor desde el inicio verás como primero mostramos en pantalla 1, después i tiene el valor 2, que sigue siendo menor que 10 por lo que se ejecuta la instrucción de mostrar i en pantalla. Después tiene el valor 3, y así sucesivamente hasta que tiene el valor 9. Una vez el valor es de nueve, lo mostramos en pantalla y le añadimos uno. En ese momento, el valor de i es de 10 por lo que la condición de ejecutar el bucle (i < 10) no se cumple y acaba el bucle. Por lo tanto en este caso mostramos desde el 1 al 9.

```
for(let i = 1; i < 10; i = i + 1) {
  console.log("El número es " + i);
}
```

En el siguiente caso, queremos mostrar desde el 1 hasta el 10. Para ello tenemos dos opciones. Podemos hacer que la condición sea i < 11 , de este modo cuando i sea 10 se ejecutará la instrucción de mostrar en pantalla. Por otro lado, podemos también poner como condición i <= 10. Una vez más, esto hará que el resultado de la condición sea true (10 <= 10) y se imprimirá el valor de 10 en pantalla.

La decisión de utilizar i < 11 o i <= 10 la debes hacer en base a la *legibilidad* del código y tu objetivo con él. Piensa si quieres que el código quiere hacer algo "*mientras el valor sea menor que 11*" o " *mientras el valor sea menor o igual que 10*" . Supongamos que nuestro programa dice cuántas cajas podemos meter en una furgoneta. En el caso de decir que hacemos el bucle mientras **i** sea menor que 11 podemos pensar que significa que cuando sean 11 cajas vamos a pedir otra furgoneta. En cambio si decimos solo podemos meter 10 cajas, por lo tanto hazlo mientras **i** sea menor o igual que 10. Al fin y al cabo la diferencia es *solamente* como tú decides explicarlo, ya que el resultado en el código será el mismo (hacerlo 10 veces).

```
for (let i = 1; i <= 10; i = i + 1) {
  console.log("El número es " + i);
}
```

En este ejemplo, en vez de sumar 1, sumaremos 2 al valor de i en cada vuelta del bucle. Ya que empezamos desde el 1, mostrará en pantalla los números impares del 1 al 10 (1, 3, 5, 7 y 9). Si quisiéramos mostrar los números pares, podríamos empezar con int i = 2;

```
for (let i = 1; i <= 10; i = i + 2) {
  console.log("El número es " + i);
}
```

Del mismo modo que podemos sumar, también podemos restar. Una vez más, ten cuidado con no crear un bucle que nunca acabe. Si empezamos con un valor en i de 1 y acabamos cuando **i** sea mayor que 10, y restamos 1 a **i** cada vez, el bucle nunca acabará sino que mostrará en pantalla 0, -1, -2, -3, -4, ... y nunca terminará. Por lo tanto, tenemos que empezar desde 10 y acabar cuando i no sea mayor que 0 (es decir, cuando i sea 0). Por supuesto, podemos utilizar los números de inicio y final que queramos, mientras la suma o resta en cada bucle lo acerque a la condición en la que acaba.

```
for (let i = 10; i > 0; i = i - 1) {
  console.log("El número es " + i);
}
```

Por supuesto, del mismo modo que sumando, podemos restar por el número que queramos, o podríamos multiplicar o dividir, el objetivo es que en algún momento, el cambio que hagamos consiga que la condición no se cumpla para poder acabar el bucle.

```
for (let i = 10; i > 0; i = i - 2) {
  console.log("El número es " + i);
}
```

Uno de los operadores que todavía no hemos visto es el denominado operador de resto (**%**). Este operador calcula el resto que queda después de dividir el número que tiene a la izquierda por el número que tiene a la derecha. Ambos números deben ser de tipo entero (número sin decimales) para que no haya resultados raros con esta operación. Supongamos por ejemplo que ponemos

```
8%3
```

Ocho entre tres sería 2 y el resto serían 2. Por lo tanto el resultado de 8 % 3 es 2.

```
6%4
```

Seis entre cuatro sería 1, y el resto sería 2. Por lo tanto el resultado de 6 % 4 es 2. Cuando hacemos la operación de %2 a distintos números vemos que pasa algo curioso.

- `2%2` es igual a 0
- `3%2` es igual a 1
- `4%2` es igual a 0
- `5%2` es igual a 1

Cuando hacemos % 2 a un número par, el resultado es de 0. Cuando lo hacemos a un número impar el resultado es de 1. Por lo tanto, si queremos saber si un número es par podemos hacer

```
if(numero % 2 == 0) {
  console.log("El número es par");
} else {
  console.log("El número es impar");
}
```

16

Puedes verlo en **16_Operador_De_Resto**.

Veamos ahora un ejemplo en el que usamos tanto el bucle de for y el operador de resto. Queremos mostrar todos los múltiplos de 6 que tenemos entre 1 y 100. Para saber si un número es un múltiplo de 6, basta con comprobar si el resto de dividir entre 6 es igual a 0. Por lo tanto para un número x, si x % 6 == 0, ese número es un múltiplo de 6.

También sabemos cómo ver todos los números desde el 1 hasta 100, sabemos que lo podemos hacer con un bucle for:

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

Por lo tanto, para mostrar en pantalla los números múltiples de 100 que hay entre 1 y 100 nos basta con:

```
for(let i = 1; i <= 100; i = i + 1) {
  if(i % 6 == 0) {
    console.log(i);
  }
}
```

Un pequeño problema con este método es que si la lista de números es muy grande, tendremos que subir hacia arriba en la pantalla mucho. Lo ideal sería que los números se pudieran escribir uno detrás de otro, con una separación de un espacio. Lo ideal sería poder decir que console.log() no introduzca un salto de línea al final, pero esto no es posible. Por lo tanto, crearemos una variable con una cadena de texto vacía (usando el símbolo para cadenas de texto y no introduciendo nada dentro, ""). De este modo, podemos utilizar la variable para ir añadiendo el texto que resulte de cada vuelta del bucle for (uniendo al valor actual de la variable **resultado** el resultado de añadir(i + " "),es decir añadirle el número y un espacio en blanco, y cuando finalice mostrar el contenido de la variable en consola.

```
let resultado = "";
for(let i = 1; i <= 100; i = i + 1) {
  if(i % 6 == 0) {
    resultado = resultado + (i + " ");
  }
}
console.log(resultado);
```

Esta vez tendremos los números en línea, separados por espacio.

17

Puedes ver ambos ejemplos y ver el resultado de uno u otro método en el proyecto **17_Operador_De_Resto**.

APARTADO FINAL

En esta última parte veremos el ejemplo más complejo de los que hemos visto, en los que combinaremos muchos de los elementos que hemos visto hasta ahora a lo largo del precurso.

18

Vamos a crear un programa que muestre en pantalla las cartas de una baraja entera. Tienes el proyecto completo en **18_Ejemplo_For_3** , pero veremos cómo lo hemos creado paso a paso. Los palos son oros, copas, espadas y bastos. Los valores de las cartas son As, 2, 3, 4, 5, 6, 7, Sota, Caballo y Rey.

Lo que queremos conseguir es mostrar los mensajes:

```
As de oros 2 de oros 3 de oros ...
Caballo de bastos Rey de bastos
```

Por supuesto, podríamos escribir todas las líneas como un console.log() por línea. Pero para tener que escribir menos código, vamos a hacerlo con bucles for. Más específicamente, vamos a hacerlo con bucles anidados. Veámoslo en pseudocódigo (mezcla entre código y lenguaje normal)

```
for( // cada uno de los palos ) {
  for( //cada uno de los valores ) {
    // mostrar el nombre de cada carta
  }
}
```

Rotamos entre oros, copas, espadas y bastos. Dentro de cada uno de ellos rotamos por los valores que puede tener cada carta. Combinándolos al mostrar la carta, podremos tener un mensaje específico por cada carta. Veamos cómo funciona:

```
for(let palo = 1; palo <= 4; palo = palo + 1) {
  for(let valor = 1; valor <= 10; valor = valor + 1) {
    // Mostrar el nombre de la carta }
  }
}
```

Por lo tanto nos faltaría mostrar el valor de cada carta. A la hora de mostrar el valor de cada carta, queremos combinar tanto el palo como el valor dentro de una cadena de texto. Una vez combinadas, lo mostraremos con un log. En primer lugar, tenemos la variable palo que nos indica el palo (1 sería Oros, 2 sería Copas, 3 sería Bastos y 4 sería Espadas). Podemos por lo tanto utilizar if para ver el valor de palo y añadir el mensaje adecuado a la cadena de texto.

```
if(palo == 1) {
  nombreCarta = nombreCarta + "Oros";
}
if(palo == 2) {
  nombreCarta = nombreCarta + "Copas";
}
if(palo == 3) {
  nombreCarta = nombreCarta + "Espadas";
}
if(palo == 4) {
  nombreCarta = nombreCarta + "Bastos";
}
```

De este modo, compararemos el valor de palo 4 veces. Esto no es necesario, ya que si es uno de ellos no será ninguno de los otros. Por lo tanto podemos utilizar else. Para poner varios if...else seguidos, podemos utilizar el formato

```
if(...) {
...
}
else if(...) {
...
}
```

Esto haría que en cuanto el valor de palo coincida, no compruebe el resto de valores posibles (también podíamos haberlo hecho con un switch).

```
if(palo == 1) {
  nombreCarta = nombreCarta + "Oros";
 }else if(palo == 2) {
  nombreCarta = nombreCarta + "Copas";
}else if(palo == 3) {
  nombreCarta = nombreCarta + "Espadas";
}else if(palo == 4) {
  nombreCarta = nombreCarta + "Bastos";
}
```

Haremos algo parecido con el valor de la carta. La variable valor puede tener entre 1 a 10. Si es entre 2 y 7, podemos simplemente escribir el número (nombreCarta = valor). En cambio si es un 1 queremos que escriba "As", un 8 "Sota", un 9 "Caballo" y un 10 "Rey". Volveremos a utilizar los if...else, pero esta vez si vemos que el valor es 1, 8, 9 o 10 escribiremos un texto específico y si no simplemente escribiremos el valor:

```
if(valor == 1) {
  nombreCarta = "As";
} else if(valor == 8) {
  nombreCarta = "Sota";
} else if(valor == 9) {
  nombreCarta = "Caballo";
} else if(valor == 10) {
  nombreCarta = "Rey";
} else {
  nombreCarta = valor;
}
```

Finalmente, entre el valor y el palo, queremos escribir " de ". Simplemente lo añadimos entre los if...else del valor y los if...else del palo.

Puedes ver el programa completo en el proyecto **18_Ejemplo_For_3**.

Ejercicio tablero

Como ejercicio final, crea un nuevo proyecto y escribe un programa que muestre en consola un "tablero":

```
OXOXOXOX
XOXOXOXO
OXOXOXOX
XOXOXOXO
OXOXOXOX
XOXOXOXO
OXOXOXOX
XOXOXOXO
```

El tamaño no es fijo, sino que le pedirás al usuario que introduzca un número y usarás ese número para tanto la altura como la anchura.

Este ejercicio va a requerir no solamente que uses lo aprendido hasta ahora, sino que también tendrás que pensar cómo aplicar lo aprendido de un modo diferente. Si no se te ocurre cómo abordar el problema, divídelo en partes más simples. ¿Sabes cómo conseguir que muestre una sola línea con ese patrón (OXOX) con el número de letras que el usuario introduzca? ¿Puedes hacer dos líneas iguales una debajo de otra? ¿Cómo podrías hacerlas distintas? **Ser capaz de ver cómo dividir un problema grande en varios fáciles es parte de lo que harás como programador.** Por supuesto si ves que necesitas ayuda, ya sabes cómo ponerte en contacto con nosotros.

Cuando acabes con estos ejercicios, ponte en contacto con cualquiera de los instructores (aquí tienes como) y te enviaremos la siguiente serie de ejercicios para que continúes en tu camino de aprendizaje de programación.

Por supuesto, y como siempre, si en algún momento te surge cualquier tipo de problema, no dudes en contactarnos por los mismos canales.