JAVASCRIPT

CLASE 2

Material complementario

CONTROL DE FLUJO

CODER HOUSE

JavaScript: condicionales

Condicionales: definición

Los condicionales son instrucciones especiales, también llamadas sentencias, que nos permiten determinar si se ejecuta o no cierta parte del programa. Es decir, son elementos que se usan para hacer o no hacer algo, teniendo en cuenta cierta situación.

En la vida cotidiana, estamos rodeados/as de condicionales, por ejemplo cuando actuamos en consecuencia al clima:

- Si hace frío, entonces me abrigo.
- Si está lloviendo, entonces llevo un paraguas.
- Si hace calor, entonces no uso un abrigo

Notemos que en los ejemplos anteriores estamos evaluando siempre el clima, y teniendo en cuenta su estado, hacemos una cosa u otra. De forma similar funcionan los condicionales en la programación: al evaluar si un variable coincide o no con cierto valor, se interpreta (o no) un conjunto de instrucciones.

Estructura if

Existen distintos condicionales que podemos emplear en JavaScript. Estos suelen identificarse bajo el nombre de *estructuras condicionales*; la más empleada es if, que se utiliza para ejecutar un bloque si la condición es verdadera. Veamos un ejemplo de declaración de esta instrucción:

```
if (true){
    console.log("vas a ver este mensaje");
}
```

Analicemos ahora los elementos que componen a esta estructura:

- **if** es la palabra reservada que inicia la instrucción,
- Los paréntesis **()** incluyen la comparación a evaluar: si lo que está entre paréntesis es verdadero (true), se interpretan las instrucciones que "están dentro" del condicional.
- Las llaves **{}** determinan qué líneas de código forman parte del condicional, y se interpretarán si la condición es verdadera. Al conjunto de instrucciones definidas entre las llaves se lo llama *bloque*. Así, es correcto decir que si la comparación es verdadera, entonces el bloque se interpreta. Caso contrario, si la comparación a evaluar resulta falsa, las instrucciones que conforman el bloque no formarán parte de la ejecución actual del programa. Como por ejemplo:

```
if (false){
    console.log("no vas a ver este mensaje");
}
```

Tenemos que tener presente que en los ejemplos anteriores lo que está dentro de los paréntesis del if no es una comparación propiamente dicha, ya que para esto deberíamos tomar una variable, y preguntarnos si es igual a un valor, como en el siguiente código:

```
let unNumero = 5

// Con (unNumero == 5) comparamos si unNumero es igual a 5
if (unNumero == 5){
    console.log("vas a ver este mensaje");
}

// Con (unNumero == 6) comparamos si unNumero es igual a 6
if (unNumero == 6){
    console.log("no vas a ver este mensaje");
}
```

Dado que el bloque que se interpreta es aquel cuya comparación es verdadera, en este caso la única salida que se visualizará por consola es la que cumple que unNumero es igual a 5.

Estructura if... else

Si bien la estructura if nos permite definir un conjunto de instrucciones a interpretarse cuando algo se cumple, puede ser necesario definir un código adicional en aquellos casos en que el resultado no es el esperado. Para estas situaciones, contamos con la estructura *if...else*, la cual nos permite contar con dos bloques, como podemos ver el siguiente ejemplo:

```
let unColor = "Rojo"

// Con (unColor == "Rojo") comparamos si unColor es igual "Rojo"
if (unColor == "Rojo"){
    console.log("el color es Rojo");
}else{
     //La instrucción se interpreta cuando unColor NO es "Rojo"
    console.log("el color NO es Rojo");
}
```

El primer bloque se ejecuta cuando unColor es igual a “Rojo”. Si unColor cambia su valor durante el funcionamiento del programa, por ejemplo la variable toma el valor de “Verde”, el bloque a interpretar es el segundo, el cual se encuentra definido luego de la palabra reservada entre llaves else.

Cabe aclarar que en las estructuras condicionales, cuando se detecta que una condición se cumple, el navegador no interpreta el resto de la estructura. Es decir, si se

cumple que unColor es igual a “Rojo”, se ingresa al primer bloque y el else nunca será leído por el intérprete.

Un caso típico de uso de if...else es para verificar si una entrada ingresada por el usuario está vacía:

```
let nombreUsuario = prompt("Ingresar nombre de usuario");

if (nombreUsuario == "") {
    alert("No ingresaste el nombre de usuario");
}
else {
    alert("Nombre de usuario ingresado " + nombreUsuario);
}
```

Dos comillas simples o dobles, sin ningún caracter entre medio, representan una cadena de caracteres vacíos. Recordemos de la clase anterior qué prompt() siempre nos otorga el valor ingresado por el usuario en tipo string. Si el usuario no escribió nada, la cadena es igual a “”, ejecutándose el primer bloque; en cambio, si ingresa al menos una letra , el bloque a interpretar es el segundo.

Estructura if...else if

Por el momento, los ejemplos mostrados de la estructura if y la estructura if...else trabajan con un único condicional. Pero, ¿qué opciones tenemos si queremos comparar una variable con más de un valor y establecer un bloque de instrucciones en respuesta para cada caso? Una opción es emplear la estructructura if...else if, como veremos a continuación:

```
let precio = 100.5;

if (precio < 20) {
    alert("El precio es menor que 20");
}
else if (precio < 50) {
    alert("El precio es menor que 50");
}
else if (precio < 100) {
    alert("El precio es menor que 100");
}
else {
    alert("El precio es mayor que 100");
}
```

Tal como se muestra en el ejemplo, es posible comparar la variable precio con más de un valor. Cuando se determina que una de las condiciones es verdadera, se interpreta dicho bloque y el resto de la estructura se omite. Si ninguna condición se cumple, se ejecuta el bloque del else por defecto, como en este caso, ya que 100.5 es mayor que 100, y no serían verdaderas ninguna de las comparaciones.

Es importante identificar que todas las instrucciones se interpretan de arriba hacia abajo, y de izquierda a derecha. En consecuencia, **el orden de los condicionales puede condicionar el bloque al que se ingresa primero**.

Por ejemplo, si ahora la variable precio es igual a 10 y asociamos al primer condicional la comparaciòn (precio < 50), ingresaríamos a dicho bloque ya que 10 es menor que 50,

pero hay que tener en cuenta que 10 es también menor que 20 y 100, pudiendo ingresar a cualquier bloque; no obstante, como ya se ingresamos al primer condicional, el resto de la estructura se omite. En este sentido, es importante que el/la programador/a defina el orden de los condicionales, teniendo presente cómo interpretan las estructuras condicionales, y qué bloque se busca ejecutar realmente.

Variable booleana

Una variable booleana es aquella que sólo tiene dos valores: es verdadera (true) o falsa (false). Identificamos a los valores true y false como *valores booleanos*.

Es común emplear variables con valores de este tipo con la intención de usarlas junto a los condicionales. Podemos identificar como definir variables booleanas en el ejemplo a continuación:

```
let esValida = true;
let numero   = 10;
let esMayor5 = (numero > 5); // su valor sera true

if (esValida) {
    alert("Es boolean true");
}
```

Se asume que si una variable booleana es verdadera, en algún momento, ya sea por alguna acción realizada por el usuario o en respuesta a alguna operación, cambiará su valor a falso, estableciendo así un cambio de flujo en la ejecución del programa.

Operadores en JavaScript

¿Qué son los operadores?

Los operadores en JavaScript son elementos que nos permiten evaluar valores y/o variables. Los usamos para construir comparaciones que podemos emplear en los condicionales. La tabla que se muestra a continuación nos permite identificar algunos operadores existentes en el lenguaje:

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

1. **Operadores de igualdad (==) y (===):** sirven para preguntarnos si dos valores o variables son iguales. No obstante, hay que tener en cuenta que en JS se manejan dos tipo de igualdades: una parcial, cuando los valores coinciden independientemente del tipo de dato (así 1 == “1” es verdadero, no importa si 1 en un número y “1” es un string, al tener el mismo valor la igualdad se cumple”) y otra estricta, la cual es verdadera cuando los valores comparados son idénticos en valor y tipo de dato (es decir, sólo 1 === 1 es verdadero).
2. **Operadores de desigualdad (!=) y (!==):** son útiles para determinar que una variable no tiene cierto valor (por ejemplo, para saber que unNumero no es igual a cero). De la misma forma que la igualdad, la desigualdad puede ser parcial o estricta, y en consecuencia 1 != "1" no son distintos, ya que tienen el mismo valor:

pero 1 !== "1" si son distintos porque tienen el mismo valor pero diferente tipo de dato.

3. **Operadores de orden (<),(<=), (>), (>=):** su utilidad principal es determinar que un valor numérico es menor o mayor que otro (por ejemplo, 5 < 6 es verdadero pero 8 < 6 es falso). Si queremos incluir al número contra el que estamos comparando, debemos emplear menor o igual, y mayor o igual (en este caso sería 8 <= 8 y 2 >= 2, ambas son verdaderas).
4. **Operadores lógicos (&&), || , (!):** utilizamos los operadores lógicos para comparar valores y/o variables booleanas. En este caso, el resultado de dichas comparaciones sigue las siguientes reglas según el operador:
    - Si usamos && (AND) para comprar dos valores booleanos, la comparación es verdadera sólo si ambos valores son verdaderos, Ejemplo: true && true es verdadero (único caso true al usar el operador AND).
    - Si usamos || (OR) para comparar dos valores booleanos, la comparación es falsa si ambos valores son falso, Ejemplo: false || false es falso (único caso false al usar el operador OR)
    - Si usamos ! (NOT) en un valor y/o variable booleana, invertimos su valor de verdad, es decir que si el valor es verdadero, con ! lo transformamos en falso, y si es falso, con ! lo transformamos en verdadero, Ejemplo: !true es falso, y !false es verdadero.

Condiciones compuestas con && (AND)

Así como podemos comparar valores booleanos con operadores lógicos, también es posible utilizarlos para construir condiciones compuestas, las cuales nos permiten codificar estructuras condicionales con la potencialidad de comparar más de un valor en una expresión. A continuación veremos un ejemplo de una condición compuesta empleando el operador && AND:

```
let nombreIngresado   = prompt("Ingresar nombre");
let apellidoIngresado = prompt("Ingresar apellido");

if((nombreIngresado !="") && (apellidoIngresado !="")){
    alert("Nombre: "+nombreIngresado +"\nApellido: "+apellidoIngresado);
}else{
    alert("Error: Ingresar nombre y apellido");
}
```

El ejemplo anterior permite determinar si las dos entradas solicitadas al usuario (nombreIngresado y apellidoIngresado) fueron cargadas.

Recordemos que && sólo es verdadero si se cumplen ambas comparaciones, y en este caso nos estamos preguntando si los valores ingresados son distintos de una cadena vacía (“”). Si el usuario ingresa al menos un caracter en ambas entradas, la condición es verdadera; por el contrario, si al menos una de las comparaciones es falsa, la condición siempre será falsa.

Recordemos que las instrucciones se interpretan de izquierda a derecha, y en el caso de condiciones compuestas, si el intérprete identifica que se emplea && (AND) y el resultado de la primera comparación es falsa (false) , no interpreta la siguiente instrucción en este caso porque sabe que la condición resultante es falsa.

Condiciones compuestas con || (OR)

También podemos utilizar el operador lógico OR para construir condiciones compuestas. A continuación, un ejemplo de dicha implementación:

```
let nombreIngresado   = prompt("Ingresar nombre");

if((nombreIngresado == "ANA") || (nombreIngresado =="ana")){
    alert("El nombre ingresado es Ana");
}else{
    alert("El nombre ingresado NO ES Ana");
}
```

Recordemos que en JavaScript cuando comparamos una letra con mayúscula y minúscula, el resultado es falso por más que se trate de la misma (por ejemplo: "a" == "A" es falso porque la minúscula es distinta que A mayúscula en valor). Entonces, podemos usar OR para armar condiciones compuestas, que determinan si el nombre ingresado está todo en minúscula o en mayúscula.

Con OR basta que una comparación sea verdadera (true) para que todo sea verdadero. La condición no se cumpliría si el usuario no ingresa “ANA” ni “ana”.

Tengamos en cuenta que las instrucciones se interpretan de izquierda a derecha, y en el caso de la condiciones compuestas, si el intérprete identifica que se emplea || (OR) y el resultado de la primera comparación es verdadero (true), no interpretará la siguiente instrucción, porque sabe que la condición resultante es verdadera.

Combinaciones AND y OR

También es posible combinar AND y OR para construir condicionales cada vez más específicos. El ejemplo que veremos a continuación valida que el nombre ingresado no sea vacío, y se escriba “EMA” o “ema”:

```
let nombreIngresado   = prompt("Ingresar nombre");

if((nombreIngresado !="") && ((nombreIngresado == "EMA") || (nombreIngresado =="ema"))){
    alert("Hola Ema");
}else{
    alert("Error: Ingresar nombre valido");
}
```

Analizemos cómo se interpreta esta estructura condicional:

1. En primer lugar, nombreIngresado tiene que tener **al menos un caracter**, dado que si la cadena es vacía, no se interpreta el resto del condicional, porque hay un && (AND), y el primer resultado de la comparación tiene que ser si o si verdadero para que exista la posibilidad de que la condición sea verdadera.
2. Luego del && (AND), encontramos dos comparaciones combinadas por un || (OR): basta con que se cumpla un caso, es decir, que el usuario escriba "EMA" o "ema", para que el condicional sea verdadero. Así, si se detecta que el usuario ingresó "EMA", el intérprete no lee la siguiente comparación dado que es suficiente con que un valor sea verdadero.
3. Los paréntesis en la condición determinan cómo el intérprete debe evaluar las condiciones. Agrupamos las comparaciones con la intención de que se lean correctamente. En este caso, funciona correctamente la verificación, pero si agrupamos las comparaciones de la siguiente manera:

```
if(((nombreIngresado !="") && (nombreIngresado == "EMA")) || (nombreIngresado =="ema")){
```

El resultado no sería el esperado, ya que en este caso con que el usuario ingrese "ema" todo el condicional sería verdadero, y se estaría haciendo un correcto control del valor vacío.