clase n°11
JavaScript

<animate a/> programar
revøluciøn* digital_

operadores en javascript

Truco para completar las etiquetas, solo escribimos el nombre de la etiqueta y presionamos TAB, clic en la Flecha.

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

```
let numberone=120;
let numbertwo=20;
let result = numberone + numbertwo;
console.log(result);
```

Comprobemos el funcionamiento de los distintos operadores.

Ahora agreguemos aclaraciones:

```
let numberone=120;
let numbertwo=20;
let resultsuma = numberone + numbertwo;
let resultresta = numberone - numbertwo;
let resultmulti = numberone * numbertwo;
let resultdivision = numberone / numbertwo;
console.log("Suma:",resultsuma);
console.log("Resta:",resultresta);
console.log("Multiplicación",resultmulti);
console.log("División",resultdivision);
```

Comparemos los dos números ingresados:

```
let comp = numberone > numbertwo;
console.log(comp);
```

Comparemos los demás operadores relacionales:

Concatenar

Unir dos o más cadenas de texto en una sola.

```
let name = ”Diego”;
let surname = ”Castiglioni”;
```

```
console.log(name + ” ” + surname);
```

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

Diferencias entre == y === en Javascript

```
1 var num = 0;
2 var str = "0";
3
4 console.log(num == str); // true
5 console.log(num === str); // false
```

Comprobar su funcionamiento.

```
var num = 0;
var str = "0";
console.log("Igual ", num==str);
console.log("Estrictamente Igual:", num===str);
```

condicionales en javascript

```
const mayorEdad = 18;
if(mayorEdad >= 18) {
    console.log("Es mayor de edad");
}else {
    console.log("Es menor de edad");
}
```

Hasta ahora hemos visto código que se ejecuta línea a línea, una detrás de otra. Pero a veces se hace necesario romper esa secuencia y crear ramas que nos permitan tomar diferentes caminos en el código dependiendo de ciertas condiciones. Por ejemplo, si permitimos el acceso a un usuario a nuestro sitio o si es mayor de edad. Si es mayor de edad tiene acceso, pero si es mayor debería acceder. A este concepto se le conoce como condicionales

Comparemos la entrada de datos de un usuario con lo guardado en una base de datos:

```
let password = 'boca2022';
let input = 'boca2022';
if (password== input){
  console.log('Login Correcto');
}
```

Probar realizar una comparación con mas de dos opciones, ejemplo: Si tu nota es menor a 3 aplazado, entre 4 y 6 desaprobado, entre 7 y 9 Aprobado y 10 Sobresaliente.

```
let nota = 3;
if (nota < 4){
  console.log('Aplazado');
} else if (nota < 7) {
  console.log('Desaprobado');
} else if (nota < 9) {
  console.log('Aprobado');
}else {
    console.log('Sobresaliente');
  }
```

Comparemos la entrada de datos de un usuario para saber si es mayor de edad:

```
var edad = 18;
if (edad >= 18){
  console.log('Sos mayor de edad');
}
```

Cambiar la edad y probar agregar si es menor

```
var edad = 17;
if (edad >= 18){
  console.log('Sos mayor de edad');
  }
else {
    console.log('Sos Menor');
}
```

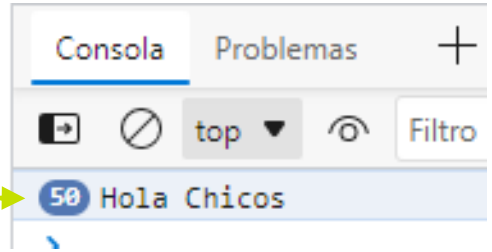
» Uso de iteradores y generadores en JavaScript para optimizar código

WHILE

Los iteradores y generadores son características adicionales, ayudan a limpiar el código y mantenerlo organizado. Mantener la lógica de iteración con el objeto al que pertenece es una buena práctica, que parece ser gran parte del enfoque de las características de los lenguajes. El estándar parece estar avanzando hacia la capacidad de estructura y facilidad de diseño de JavaScript, al tiempo que mantiene la velocidad de desarrollo de lenguajes dinámicos.

Estos editores resaltan texto con diferentes colores para ayudarnos a identificar cada parte del código, o listan los archivos de un proyecto en un panel lateral para ayudarnos a trabajar con múltiples archivos al mismo tiempo. La siguiente es una lista de los editores más populares:

En la consola, si tenemos el mismo resultado en la salida, los enumera.




```
let count = 50;

while(count > 0){
console.log("Hola Chicos");
count = count -1;
}
```

Aquí vemos como nos muestra un mensaje por cada Estudiante

```
let count = 50;

while(count > 0){
console.log("Hola Estudiante nº", count);
count = count -1;
}
```

```
let count = 50;

while(count > 0){
console.log("Hola Estudiante nº", count);
count = count -1;
}
```

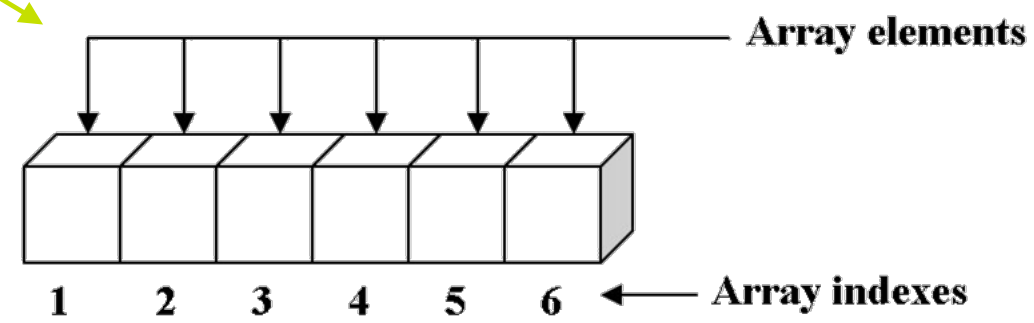
C++ o C--

FOR

Se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute

```
let nombre=['Diego','Anabel','Esteban'];
console.log(nombre);
console.log(nombre[0]);
console.log(nombre[2]);
console.log("Cantidad de Nombre:", nombre.length);
for (var i = 0; i < nombre.length; i++) {
  console.log(nombre[i]);
}
```

Un array o matriz es simplemente es una variable que puede contener valores múltiples, a diferencia de una variable regular que solo puede contener un único valor.



Armar un arreglo con frutas

```
const frutas = ['🍅', '🍄', '🥦', '🥒', '🌽', '🥕', '🥑'];
for(let i=0; i<frutas.length; i++) {
  console.log("El elemento es:",  frutas[i]);
}
```

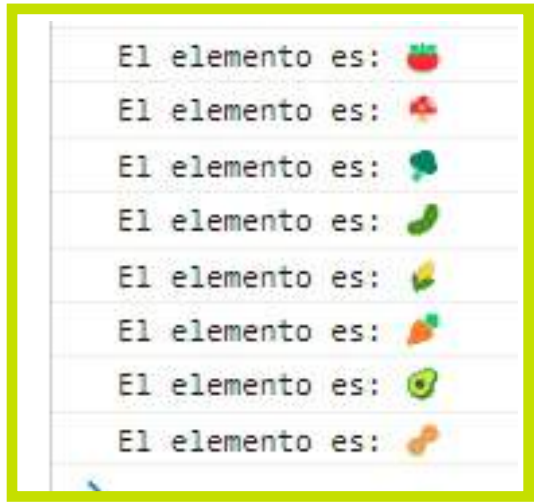
```
El elemento 🍅
El elemento 🍄
El elemento 🥦
El elemento 🥒
El elemento 🌽
El elemento 🥕
El elemento 🥑
>
```

Como añadir elementos al arreglo en JS

Usa el método **push()** para añadir un elemento en el arreglo. El método **push()** añade un elemento al final del arreglo. Ve como añadimos algunos cacahuetes a la ensalada, como esto:

```
salad.push('🥜');
```

Nota que el método **push()** añade un elemento al final del arreglo. Si tu quieres añadir un elemento al inicio del arreglo, vas a necesitar usar el método **unshift().**



Como eliminar elementos de un arreglo en JS

La manera más sencilla de eliminar un solo elemento de un arreglo usando el método **pop()** . Cada vez que llamas el método **pop()**, este elimina un elemento del final de un arreglo. Entonces este regresa con el elemento eliminado y cambia el arreglo original.

PROPIEDADES DE ARREGLOS:

- `push()` – Inserta un elemento al final del arreglo.
- `unshift()` – Inserta un elemento al inicio del arreglo.
- `pop()` – Remueve un elemento del final del arreglo.
- `shift()` – Remueve un elemento del principio del arreglo.
- `slice()` – Crea una copia sombra del arreglo.
- `length` – Determina el tamaño del arreglo.

Switch Case

El condicional Switch case es una estructura que evalúa más de un caso y se caracteriza por: Selección de una opción entre varias. Switch recibe un “caso” y lo evalúa hasta encontrar el caso que corresponda. Se puede usar la opción “default” para cuando no se encuentra el caso dado.

```
let expr = "Adios";
switch (expr) {
  case 'Hola':
    console.log("Buenos Dias");
    break;
  case 'Adios':
    console.log("Hasta Pronto");
    break;
  case 'Ayuda':
    console.log("En que te puedo Ayudar?");
    break;
  default:
    console.log("Disculpa no te entiendo");
}
```

Imaginemos que promovemos el desafío de realizar ejercicio durante la semana:

```
let ejercicio ="Lunes";
switch (ejercicio) {
  case 'Lunes':
    console.log("Caminata de 5 km");
    break;
  case 'Martes':
    console.log("Caminata de 10 km");
    break;
  case 'Miercoles':
      console.log("Correr 5 km");
    break;
  case 'Viernes':
    console.log("Correr de 10 km");
    break;
  default:
    console.log("Descanso");
}
```

revølución*
digital_