

# Arreglos

Los arreglos no son más ni menos qué un tipo especial de variable qué puede contener más de un valor al mismo tiempo

## Qué valores pueden contener?

Los mismos valores que vimos anteriormente. Es decir, pueden contener variables numéricas, string o booleanas. También es importante entender que no necesariamente deben ser todas del mismo tipo. Por ejemplo un arreglo puede contener una variable tipo string y otra numérica.

## Sintaxis

Para trabajar con arreglos lo podremos hacer de la siguiente forma

```
<script>

var arreglo = ['juan', 'marcelo', 20, 40]

</script>
```

También se puede trabajar de la siguiente manera, aunque por supuesto menos utilizada qué la anterior,

```
<script>

var arreglo = new Array('juan', 'marcelo', 20, 40);

</script>
```

Recomendamos el modo número 1 , ya que es más rápido y simple.

## Cómo mostrar los datos de un arreglo?

Por supuesto, es importante mostrar los datos contenidos en un array. Lo haremos de la siguiente manera

```
<script>

var arreglo = ['juan', 'marcelo', 20, 40];
alert(arreglo[0]);

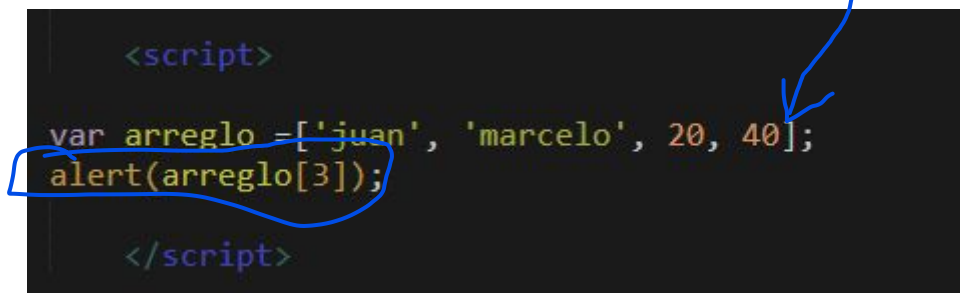
</script>
```

Es decir que siempre identificamos al dato, por encontrarse en tal o cual posición del arreglo. Por ejemplo siempre el índice del primer elemento es 0, por lo tanto si quiero mostrar el número 40(cuarenta) haré lo siguiente,

```
<script>

var arreglo = ['juan', 'marcelo', 20, 40];
alert(arreglo[3]);

</script>
```

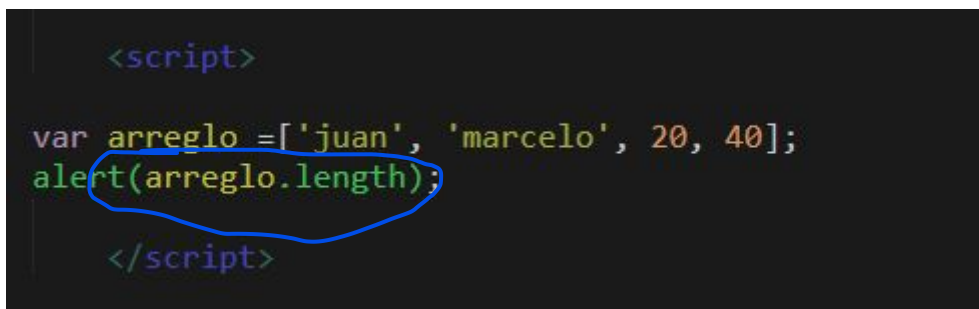


Claro está que también puedo obtener la cantidad de datos que hay un arreglo, por caso,

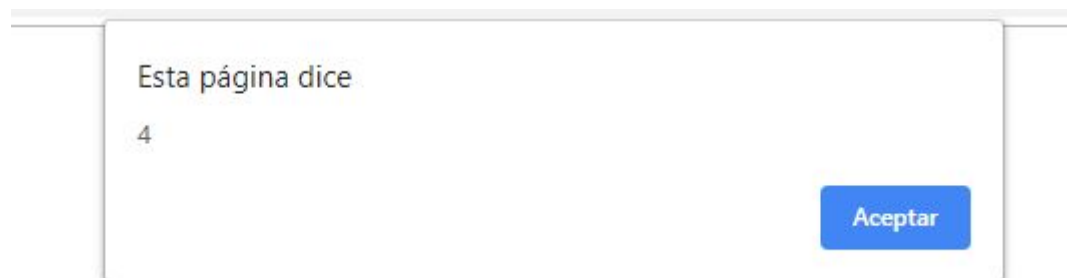
```
<script>

var arreglo = ['juan', 'marcelo', 20, 40];
alert(arreglo.length);

</script>
```



El resultado será el siguiente,



Es decir que en mi arreglo poseo 4 datos. Esto es interesante también, no sólo para saber cuántos datos hay en el arreglo sino también para cuándo itero (lo recorro).

Si quisiera mostrar absolutamente todos los datos del arreglo al mismo tiempo, lo haría de la siguiente forma

```
<body>

  <div id="caja"> </div>

  <script>

var arreglo =['juan', 'marcelo', 20, 40];
document.getElementById("caja").innerHTML = arreglo;

  </script>

</body>
```

De esta manera, el resultado en el contenedor cuyo id es caja sería el siguiente,

juan,marcelo,20,40

Vamos a suponer que estamos trabajando con un arreglo donde mencionamos los nombres de los empleados de una empresa,

```
<script>

var arreglo =['juan', 'marcelo', 'ana', 'luis'];

</script>
```

Luego mostramos esa información en un contenedor de nuestra interfaz,

```
<div id="caja"> </div>

<script>

var arreglo=['juan', 'marcelo', 'ana', 'luis'];
document.getElementById('caja').innerHTML = arreglo

</script>
```

El resultado sería el siguiente,

juan,marcelo,ana,luis

Ahora, qué pasaría si también tuviéramos el apellido de juan, o la edad de luis, o la dirección de ana?, deberíamos generar otro arreglo aparte?

## Objetos

Los objetos pueden contener propiedad y valores, por lo tanto nos son más sencillos para poder indagar características de un elemento más complejo que las variables consideradas anteriormente. Por ejemplo,

```
<script>

var alumno = { nombre: 'Ivan', apellido: 'García'}

</script>
```

## Sintaxis

Si queremos mostrar la información del objeto anterior llamado alumno, haremos lo siguiente,

```
<script>
var alumno = { nombre: 'Ivan', apellido: 'García'}
alert(alumno)
</script>
```

objeto

El resultado será el siguiente,

Esta página dice  
[object Object]

Aceptar

El problema fundamental, es que estamos trayendo al objeto entero y necesitamos en todo caso mostrar partes del mismo , por ejemplo,

```
<script>
var alumno = { nombre: 'Ivan', apellido: 'García'}
alert(alumno.nombre)
alert(alumno.apellido)
</script>
```

El resultado en el primer caso será el siguiente,

Esta página dice  
Ivan

Aceptar

Y en el segundo caso

Esta página dice

García

Aceptar

## Arreglos + Objetos

Entendiendo el trabajo tanto con arreglos como con objetos, podemos integrar ambos conceptos de la siguiente manera. Tenemos un listado de empleados. Estos empleados tienen diferentes propiedades, pero a su vez son varios y queremos tenerlos en una misma variable (arreglo), lo que haremos será lo siguiente,

```
<script>
var empleados = [{nombre: 'Juan', apellido : 'Pedro',
edad: 40}, {nombre: 'Ana', apellido : 'García',
edad: 25}, {nombre: 'Maria', apellido : 'Zarate',
edad: 34}]

</script>
```

De esta manera podemos organizarnos de mejor forma y trabajar sencillamente así como rápido con nuestro listado de empleados.

## Métodos útiles para trabajar con Arrays

Otra cosa que también podemos hacer, es utilizar algunos métodos útiles para trabajar con arreglos. Por ejemplo,

```
empleados.sort()
```

De esta manera podemos ordenar nuestro arreglo, también podemos acceder rápidamente al último dato del arreglo, por ejemplo

```
var ultimo = empleados[empleados.length - 1];
```

Para luego mostrarlo en pantalla en una alerta,

```
var ultimo = empleados[empleados.length - 1];  
alert(ultimo)
```

Por supuesto nos encontraremos nuevamente con el mismo problema planteado anteriormente,

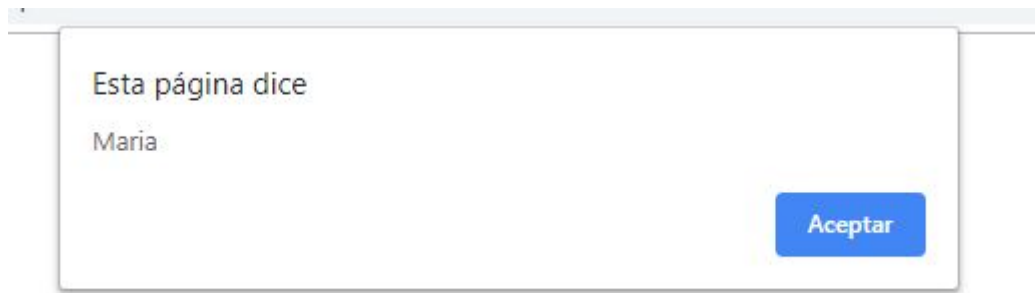
Esta página dice  
[object Object]

Aceptar

De esta manera podemos hacer lo siguiente,

```
<script>  
var empleados = [{nombre: 'Juan', apellido : 'Pedro',  
  edad: 40}, {nombre: 'Ana', apellido : 'García',  
  edad: 25}, {nombre: 'Maria', apellido : 'Zarate',  
  edad: 34}]  
  
var ultimo = empleados[empleados.length - 1];  
alert(ultimo.nombre)  
  
</script>
```

De esta forma el resultado será el siguiente,



## Push

Podemos agregar un dato al final del arreglo, por ejemplo, generamos el siguiente arreglo,

```
var empleados = ['Juan', 'Roberto']
```

Para agregar un dato al final, lo haremos de la siguiente forma,

```
<script>
var empleados = ['Juan', 'Roberto']
empleados.push('Maria')

</script>
```

Luego lo mostraremos en una caja,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto']

empleados.push('Maria')

document.getElementById('caja').innerHTML = empleados

</script>
```



El resultado en nuestro navegador, será el siguiente

Juan,Roberto,Maria

## Pop

Si queremos sacar un dato de nuestro arreglo, lo haremos con pop()

```
<script>
var empleados = ['Juan', 'Roberto']

empleados.pop()

document.getElementById('caja').innerHTML = empleados

</script>
```

El resultado será el siguiente,

Juan

## Shift, unshift

También, podemos sumar un dato al principio del arreglo, por ejemplo,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto']

empleados.unshift('Maria')

document.getElementById('caja').innerHTML = empleados

</script>
```

El resultado en el navegador, será el siguiente,

**Maria**,Juan,Roberto

En este caso María se agrega al principio, no al final. Por su lado, **si quiero quitar a María** (quitar un dato del inicio, lo haré de la siguiente forma,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto']

empleados.shift()

document.getElementById('caja').innerHTML = empleados

</script>
```

De esta manera, el resultado será el siguiente,

**Roberto**

## Splice

Si no queremos quitar datos exactamente al inicio o al final, lo que podemos hacer es lo siguiente,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']
empleados.splice(2)
document.getElementById('caja').innerHTML = empleados
</script>
```

Removerá en este caso el elemento cuyo índice es 2(dos) en adelante, si queremos también indicar cuantos elementos serán removidos, agregaremos algo más,

```
<body>

  <div id="caja"> </div>

  <script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']

empleados.splice(2,1)
document.getElementById('caja').innerHTML = empleados

  </script>

</body>
```

El resultado será el siguiente,

Juan,Roberto,Ana

No sólo podemos hacer esto, sino que también podemos al mismo tiempo qué quitamos, agregar datos en esa misma posición,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']

empleados.splice(2,1, 'Carla')
document.getElementById('caja').innerHTML = empleados

</script>
```

El resultado, será equivalente a

Juan,Roberto,Carla,Ana

Es decir,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']

empleados.splice(2,1, 'Carla')
//(posición para remover, cuantos a partir de esa
//posición, datos que agrego )
document.getElementById( 'caja' ).innerHTML = empleados

</script>
```

## concat

También se pueden unir diferentes arreglos, por ejemplo, vamos a suponer que tenemos dos sectores,

```
<script>
var sectorA = ['Juan', 'Roberto', 'Maria', 'Ana']
var sectorB = ['Pedro', 'Carlos', 'Rosa', 'Carla']

</script>
```

Pero por alguna razón, sólo en algún momento preciso queremos unir ambos arreglos, lo ahremos de la siguiente forma

```
<script>
var sectorA = ['Juan', 'Roberto', 'Maria', 'Ana']
var sectorB = ['Pedro', 'Carlos', 'Rosa', 'Carla']
var sectores = sectorA.concat(sectorB);

</script>
```

Si lo queremos mostrar en un contenedor lo haremos de la siguiente forma,

```

    <div id="caja"> </div>

    <script>
    var sectorA = ['Juan', 'Roberto', 'Maria', 'Ana']
    var sectorB = ['Pedro', 'Carlos', 'Rosa', 'Carla']
    var sectores = sectorA.concat(sectorB);

    document.getElementById('caja').innerHTML = sectores
    </script>

```

El resultado, será el siguiente

Juan,Roberto,Maria,Ana,Pedro,Carlos,Rosa,Carla

## Slice

Slice nos permite partir un arreglo a partir de cierta posición, por ejemplo,

```

    <div id="caja"> </div>

    <script>
    var sectorA = ['Juan', 'Roberto', 'Maria', 'Ana']

    var chicas = sectorA.slice(2)

    document.getElementById('caja').innerHTML = chicas;
    </script>

```

De esta manera indicamos qué queremos partir el arreglo a partir de la posición 2, de esta forma el resultado será el siguiente,

Maria,Ana

## Estructuras, bucles

Los bucles pueden ejecutar una porción de código una cantidad x de veces mientras se de una condición, por ejemplo

### FOR

Vamos a suponer que queremos recorrer un arreglo de empleados, por ejemplo el siguiente,

```
<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']

</script>
```

Cómo lo podríamos hacer a partir de lo que ya sabemos,

```
<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']

mensaje += empleados[0] + "<br>";
mensaje += empleados[1] + "<br>";
mensaje += empleados[2] + "<br>";
mensaje += empleados[3] + "<br>";
mensaje += empleados[4] + "<br>";
mensaje += empleados[5] + "<br>";

document.getElementById(caja) = mensaje;
</script>
```

Un verdadero dolor de cabeza, máxime si encima tenemos una cantidad variable de datos, o muchísimos elementos que mostrar, cómo podemos hacerlo más dinámico y fácil?

### Sintaxis

El for trabaja de la siguiente manera,

```
for(var i = 0; i < 2; i++)  
alert(i)
```

El resultado será el siguiente,

Esta página dice

0

Aceptar

Para luego mostrar el siguiente número,

Esta página dice

1

Aceptar

Y parar en ese dato, pues hemos fijado que i nunca debe ser mayor que 2.

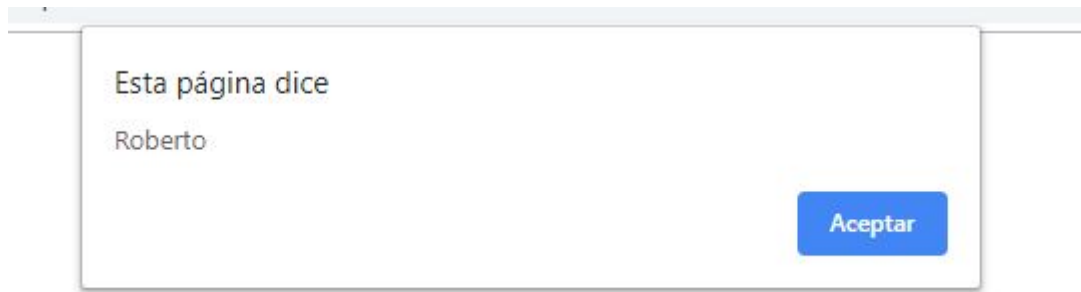
De esta manera, **lo que hacemos es repetir un ciclo (aumentar en este caso i) en tanto y en cuanto el mismo no supere a 2.**

La realidad es que este ejemplo anterior es útil, carece de sentido, a menos que lo utilizemos por ejemplo para cuestiones más prácticas como iterar un arreglo,

```
<script>  
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']  
var mensaje = "";  
  
for(var i = 0; i < 2; i++)  
  
alert(empleados[i])  
  
</script>
```



El resultado será perfecto, y nos mostrará el nombre de cada empleado siempre que no supere al índice del arreglo 2, es decir nos mostrará hasta Roberto,



Si quiero mostrar esta información en mi contenedor,

```
<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']
var mensaje = "";

for(var i = 0; i < 2; i++)
  mensaje = empleados[i]
document.getElementById('caja').innerHTML = mensaje;

</script>
```

El resultado será el siguiente,

Roberto

Claramente no es lo que esperamos, porque en realidad, el problema reside en que a diferencia de una alerta, todo el tiempo reemplazamos el contenido de la caja, y nos muestra el último valor, lo que debemos hacer es lo siguiente, utilizar un operador de asignación que ya hemos visto,

```

<body>

    <div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']
var mensaje = "";

for(var i = 0; i < 2; i++)
mensaje += empleados[i]
document.getElementById('caja').innerHTML = mensaje;

</script>

</body>

```

El resultado será el siguiente,

JuanRoberto

Claro que no queda del todo bien en la presentación, por lo tanto lo que haremos será agregar algunos elementos de HTML, por ejemplo ,

```

    <div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']
var mensaje = "";

for(var i = 0; i < 2; i++)
mensaje += empleados[i] + '<br>'
document.getElementById('caja').innerHTML = mensaje ;

</script>

```

Por otro lado, qué pasaría si en realidad, esto nos limita demasiado, y quizás ese arreglo aumenta de manera dinámica, como haremos,

```

<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana']
var mensaje = "";

for(var i = 0; i < empleados.length ; i++)

mensaje += empleados[i] + '<br>'

document.getElementById('caja').innerHTML = mensaje

</script>

```

De esta manera siempre , se adaptará la condición a la cantidad de datos que tenga en el arreglo, por ejemplo,

Juan  
Roberto  
Maria  
Ana

Si luego hago esto,

```

var empleados = ['Juan', 'Roberto', 'Maria', 'Ana', '
Pedro']
var mensaje = ""

```

Es decir agrego a Pedro, el contenedor automáticamente mostrará todos los datos del arreglo, sin necesidad que se deba variar el límite,

Juan  
Roberto  
Maria  
Ana  
Pedro

## While

El bucle while es similar, trabajando su sintaxis de la siguiente manera,

```
while (i < 10) {  
    mensaje += "El dato es " + i;  
    i++;  
}
```

De esta forma, el dato lo podemos trabajar de la siguiente manera,

```

<div id="caja"> </div>

<script>
var empleados = ['Juan', 'Roberto', 'Maria', 'Ana', '
    Pedro']
var mensaje = "";
var i = 0;
while(i < empleados.length) {

mensaje += empleados[i] + '<br>'

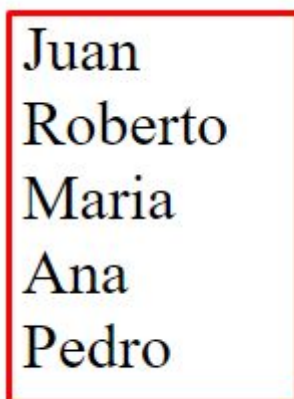
i++
}

document.getElementById('caja').innerHTML = mensaje

</script>

```

El resultado será el siguiente,



Juan  
Roberto  
Maria  
Ana  
Pedro

## Iteración de Arreglos con objetos

Si por ejemplo nuestro arreglo tiene objetos y queremos recorrerlo como hemos aprendido lo haremos de la siguiente forma

```
<div id="caja"> </div>

<script>
var empleados = [{nombre: 'Juan', apellido : 'Pedro',
edad: 40}, {nombre: 'Ana', apellido : 'García',
edad: 25}, {nombre: 'Maria', apellido : 'Zarate',
edad: 34}]

var mensaje = "";
var i = 0;
while(i < empleados.length) {

mensaje += empleados[i].nombre + '<br>'

i++
}

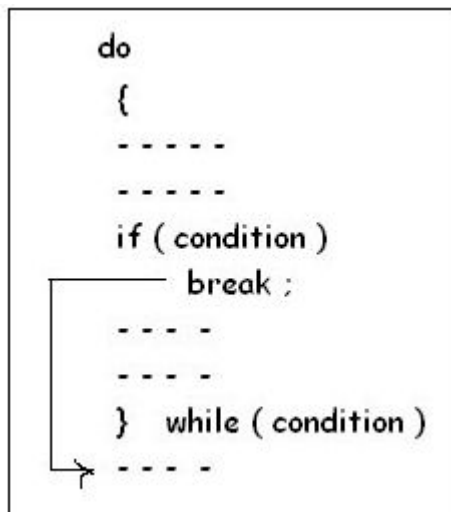
document.getElementById('caja').innerHTML = mensaje

</script>
```

## Extras: Sentencias break y continue

### Break

La sentencia break se utiliza para interrumpir un bucle antes de lo esperado.  
Suele escribirse en muchos lenguajes: **break**;



Esta sentencia es válida dentro de un bucle (cualquiera de ellos); en cambio NO es válida fuera de un bucle, lo cual generará errores en el programa. ¿Por qué interrumpir un bucle antes de tiempo? Puede darse una situación en la cual el bucle no debe finalizar “naturalmente”, sino forzar su interrupción antes, dada cierta circunstancia.

Es decir, **el uso de break dentro de un bucle siempre estará condicionado:**

```

if(condición){
    break; //Se termina la iteración
}

```

Esta sentencia es muy útil cuando se trabaja con arreglos, uno de los próximos temas.

## Continue

A diferencia de la sentencia break, continue interrumpe solamente la iteración ACTUAL, y NO todo el bucle: Al ejecutarse un continue, se procederá con la siguiente iteración. Continue siempre es válido dentro de un bucle, y nunca fuera de ellos.

Su uso dentro del bucle estará condicionado:

```

if(condición){
    continue; //Se continúa con el siguiente paso dentro del bucle si es que hubiera alguno
}

```

Por ejemplo, si quisiéramos imprimir en pantalla los números en pantalla desde el 1 al 30, excepto el 13, la sentencia continue resulta sumamente útil:

```

for(i = 1; i <= 30; i++){
    if(i == 13){
        continue;
    }
}

```

```
}  
console.log(i);  
}
```

El valor numérico de “i” siempre se imprime en pantalla, excepto cuando “i” alcanza el valor “13”: En ese caso se ejecutará la sentencia `continue`; por lo cual finalizará en ese mismo momento la iteración actual, pasando inmediatamente a iterar con `i = 14`.



