

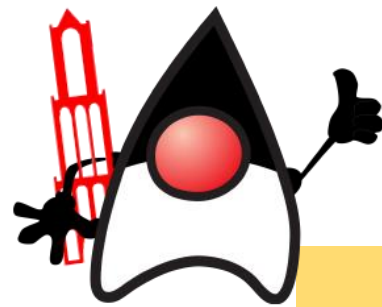
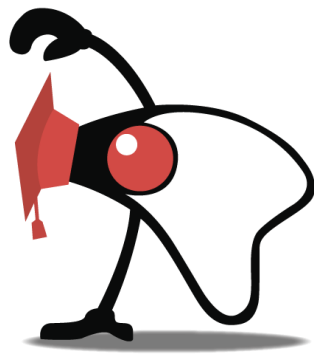


<codoa  
codo/>



# Curso FullStack Python

Codo a Codo 4.0



# JavaScript

## *Parte 5*



# Array (arreglo o vector)

Un array es una **colección o agrupación de elementos** en una misma variable, cada uno de ellos ubicado por la posición que ocupa en el array. En Javascript, se pueden definir de varias formas:

| Constructor                       | Descripción   |
|-----------------------------------|---|
| <code>new Array(len)</code>       | Crea un array de len elementos .  |
| <code>new Array(e1, e2...)</code> | Crea un array con ninguno o varios elementos.                           |
| <code>[e1, e2...]</code>          | Simplemente, los elementos dentro de corchetes: []. Notación preferida. |

```
// Forma tradicional
const array = new Array("a", "b", "c");
```

JS

```
// Mediante literales (preferida)
const array = ["a", "b", "c"]; // Array con 3 elementos
const empty = []; // Array vacío (0 elementos)
const mixto = ["a", 5, true]; // Array mixto (string, number, boolean)
```

# Array | Acceso a elementos

| Propiedad            | Descripción   |
|----------------------|---|
| <code>.length</code> | Devuelve el número de elementos del array.              |
| <code>[pos]</code>   | Operador que devuelve el elemento número pos del array. |

```
const array = ["a", "b", "c"];  
  
console.log(array[0]); // 'a'  
console.log(array[2]); // 'c'  
console.log(array[5]); // undefined  
console.log(array.length); // 3
```

JS

|           |
|-----------|
| a         |
| c         |
| undefined |
| 3         |

*Debemos pensar al vector como los vagones de un tren, donde cada vagón va a tener un contenido y un orden. El índice es el orden y el contenido transportado por el vagón del tren es el dato.*

*Las posiciones de un array se numeran a partir de **0 (cero)**. Cuando usamos **array[0]** estamos haciendo referencia a la posición 0 del array cuyo contenido, en este caso, es la letra "a".*

*En el caso de **array[5]** estamos haciendo referencia a una posición que no existe porque el array tiene 3 posiciones, mientras que **length** es una propiedad del array que devuelve la cantidad de elementos que tiene el array.*

# Array | Ejemplos (crear, acceder y mostrar elementos)

```
const vector= [1,3,5,8]; //0, 1, 2, 3: cantidad de elementos - 1
const vectorVacio= []; //Vector vacío
const vectorDos = new Array("a", "b", "c");
const vectorTres = new Array (1, 5, 10, 15, 20);
```

JS

```
console.log(vector);
document.write(vector);
console.log("Vector vacío:", vectorVacio);
console.log(vectorDos);
console.log(vectorDos[1]);
console.log(vectorTres[2]);
console.log(vectorTres[6]);
```

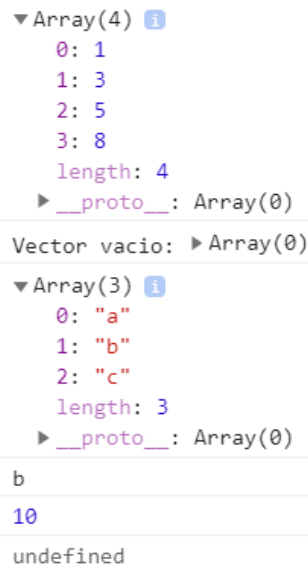
JS

1,3,5,8

**Tip:** Para acceder al último elemento del array hacemos:

`let ultimo = nombreArray[nombreArray.length - 1]`

donde agregamos - 1 porque las posiciones empiezan desde 0.



```
▼ Array(4) i
  0: 1
  1: 3
  2: 5
  3: 8
  length: 4
  ► __proto__: Array(0)
Vector vacío: ► Array(0)
▼ Array(3) i
  0: "a"
  1: "b"
  2: "c"
  length: 3
  ► __proto__: Array(0)
b
10
undefined
```

Ver ejemplo arrays  
(.html y .js)

## Array | Ejemplos (crear, acceder y mostrar elementos)

```
console.log("Elementos del vector 2:");  
for(i = 0; i < vectorDos.length; i++){  
    console.log(vectorDos[i]);  
}
```

JS

Elementos del vector 2:

a

b

c

Utilizando un **for** mostramos el vectorDos. Para ello empleamos **vectorDos.length** que nos da el largo del vector y el **< (menor que)** para recorrer la cantidad de posiciones - 1 (dado que las posiciones en el vector comienzan por 0, sino la última daría indefinido).

```
document.write("Elementos del vector 3: <br>");  
for(i = 0; i < vectorTres.length; i++){  
    document.write(vectorTres[i] + ", ");  
}
```

JS

Elementos del vector 3:  
1, 5, 10, 15, 20,

Utilizando un **for** mostramos el vectorTres de la misma manera que el vectorDos. En este caso mostramos en el body cada elemento del vector, separados por una coma.

[Ver ejemplo arrays \(.html y .js\)](#)

# Array | Métodos (funciones)

| Método                               | Descripción  |
|--------------------------------------|--|
| <code>.push(obj1, obj2...)</code>    | Añade uno o varios elementos al final del array. Devuelve tamaño del array.  |
| <code>.pop()</code>                  | Elimina y devuelve el último elemento del array.                             |
| <code>.unshift(obj1, obj2...)</code> | Añade uno o varios elementos al inicio del array. Devuelve tamaño del array. |
| <code>.shift()</code>                | Elimina y devuelve el primer elemento del array.                             |
| <code>.concat(obj1, obj2...)</code>  | Concatena los elementos (o elementos de los arrays) pasados por parámetro.   |
| <code>.indexOf(obj, from)</code>     | Devuelve la posición de la primera aparición de obj desde from.              |
| <code>.lastIndexOf(obj, from)</code> | Devuelve la posición de la última aparición de obj desde from.               |

**Más información:** <https://lenguajejs.com/javascript/fundamentos/arrays/>

*Para estos métodos ver ejemplo  
arrays-metodos (.html y .js)*

# Array | Métodos | Push, Pop, Unshift y Shift

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango"]; JS
```

## .push(obj1, obj2...)

```
frutas.push("Kiwi", "Pera"); JS
```

PUSH

Banana,Naranja,Manzana,Mango

Banana,Naranja,Manzana,Mango,Kiwi,Pera

## .pop()

```
frutas.pop(); //No tiene argumentos JS
```

POP

Banana,Naranja,Manzana,Mango,Kiwi,~~Pera~~

Banana,Naranja,Manzana,Mango,Kiwi

```
var colores = ["Rojo", "Amarillo", "Verde", "Celeste"]; JS
```

## .unshift(obj1, obj2...)

```
colores.unshift("Azul", "Naranja"); JS
```

UNSHIFT

Rojo,Amarillo,Verde,Celeste

Azul,Naranja,Rojo,Amarillo,Verde,Celeste

## .shift()

```
colores.shift(); //No tiene argumentos JS
```

SHIFT

~~Azul~~,Naranja,Rojo,Amarillo,Verde,Celeste

Naranja,Rojo,Amarillo,Verde,Celeste



# Array | Métodos | Concat, IndexOf y LastIndexOf

```
var varones = ["Juan", "Pablo"];  
var masVarones = ["Luis", "Pedro"];
```

JS

## .concat(obj1, obj2...)

```
var todos = varones.concat(masVarones);
```

JS

```
todos = todos.concat("Julieta", "Natalia");
```

JS

CONCAT  
Juan,Pablo  
Luis,Pedro  
Juan,Pablo,Luis,Pedro  
Juan,Pablo,Luis,Pedro,Julieta,Natalia

*Declaramos dos vectores y los concatenamos en un tercer vector llamado **todos**. Luego, a ese vector le incorporamos dos valores nuevos.*

```
var letras = ["A", "B", "C", "D", "E", "B", "C"];
```

JS

## .indexOf(obj, from)

```
var pos = letras.indexOf("B");  
pos = letras.indexOf("C", 4);
```

JS

B está en la posición: 1  
C está en la posición: 6

*En el segundo  
caso partimos  
desde la pos 4*

## .lastIndexOf(obj, from)

```
var pos2 = letras.lastIndexOf("B");  
pos2 = letras.lastIndexOf("B", 4); //de derecha  
a izquierda
```

JS

B está en la posición: 5  
B está en la posición: 1

*En el segundo caso parte de la  
pos 4 (desde la derecha)*

## Array | Otros métodos

| Método          | Descripción   | Referencia  |
|-----------------|---|---|
| <b>splice()</b> | El método splice () agrega / elimina elementos a / de un array y devuelve los elementos eliminados.   | <a href="https://www.w3schools.com/jsref/jsref_splice.asp">https://www.w3schools.com/jsref/jsref_splice.asp</a>           |
| <b>.slice()</b> | El método slice () devuelve los elementos seleccionados en un array, como un nuevo objeto de array.<br>Selecciona los elementos que comienzan en el argumento inicial dado y finalizan en el argumento final dado, pero no lo incluyen. | <a href="https://www.w3schools.com/jsref/jsref_slice_array.asp">https://www.w3schools.com/jsref/jsref_slice_array.asp</a> |

# Array | Métodos de orden

| Método                   | Descripción   |
|--------------------------|---|
| <code>.reverse()</code>  | Invierte el orden de elementos del array.                                 |
| <code>.sort()</code>     | Ordena los elementos del array bajo un criterio de ordenación alfabética. |
| <code>.sort(func)</code> | Ordena los elementos del array bajo un criterio de ordenación func.       |

## `.reverse()`

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango", "Kiwi", "Pera"];  
document.write(frutas, "<br>");  
document.write(frutas.reverse());
```

JS

Banana,Naranja,Manzana,Mango,Kiwi,Pera  
Pera,Kiwi,Mango,Manzana,Naranja,Banana

*Para estos métodos ver ejemplo arrays-orden (.html y .js)*

# Array | Métodos de orden

.sort()

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango", "Kiwi", "Pera"];  
document.write(frutas, "<br>");  
document.write(frutas.sort());
```

JS

Banana,Naranja,Manzana,Mango,Kiwi,Pera  
Banana,Kiwi,Mango,Manzana,Naranja,Pera

# Array | Métodos de orden

## Función de comparación

Como hemos visto, la ordenación que realiza `sort()` por defecto es siempre una ordenación alfabética. Sin embargo, podemos pasarle por parámetro lo que se conoce con los nombres de función de ordenación o función de comparación. Dicha función, lo que hace es establecer otro criterio de ordenación, en lugar del que tiene por defecto:

```
// Función de comparación para ordenación natural
const numeros = [1, 8, 2, 32, 9, 7, 4];

const asc = function (a, b) {
    return a - b;
};
const desc = function (a, b) {
    return b - a;
};
document.write(numeros);
document.write(numeros.sort(asc));
document.write(numeros.sort(desc));
```

JS

1,8,2,32,9,7,4

*Desordenado*

1,2,4,7,8,9,32

*Ascendente*

32,9,8,7,4,2,1

*Descendente*

## Función de comparación (continuación)

### Sintaxis

`array.sort(compareFunction)`

### Valores de parámetros

| Parámetro              | Descripción  |
|------------------------|--|
| <i>compareFunction</i> | <p>Opcional. Una función que define un orden de clasificación alternativo. La función debe devolver un valor negativo, cero o positivo, según los argumentos, como:</p> <pre>function(a, b){return a-b}</pre> <p>Cuando el método <code>sort ()</code> compara dos valores, envía los valores a la función de comparación y ordena los valores de acuerdo con el valor devuelto (negativo, cero, positivo).</p> <p><b>Ejemplo:</b></p> <p>Al comparar 40 y 100, el método <code>sort ()</code> llama a la función de comparación (40,100).<br/>La función calcula 40-100 y devuelve -60 (un valor negativo).<br/>La función de ordenación clasificará 40 como un valor inferior a 100.</p> |

# Array | Más métodos (funciones)

| Método                             | Descripción  |
|------------------------------------|--|
| <code>.forEach(cb, arg)</code>     | Realiza la operación definida en cb por cada elemento del array.       |
| <code>.every(cb, arg)</code>       | Comprueba si todos los elementos del array cumplen la condición de cb. |
| <code>.some(cb, arg)</code>        | Comprueba si al menos un elem. del array cumple la condición de cb.    |
| <code>.map(cb, arg)</code>         | Construye un array con lo que devuelve cb por cada elemento del array. |
| <code>.filter(cb, arg)</code>      | Construye un array con los elementos que cumplen el filtro de cb.      |
| <code>.findIndex(cb, arg)</code>   | Devuelve la posición del elemento que cumple la condición de cb.       |
| <code>.find(cb, arg)</code>        | Devuelve el elemento que cumple la condición de cb.                    |
| <code>.reduce(cb, arg)</code>      | Ejecuta cb con cada elemento (de izq a der), acumulando el resultado.  |
| <code>.reduceRight(cb, arg)</code> | Idem al anterior, pero en orden de derecha a izquierda.                |

**Más información:** <https://lenguajejs.com/javascript/caracteristicas/array-functions/>

*Para estos métodos ver ejemplo arrays-metodos2 (.html y .js)*

# Array | Más métodos | forEach y every

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango"];
```

JS

## .forEach(cb, arg)

```
frutas.forEach(mostrar);  
function mostrar(elemento, indice) {  
    document.write(indice + ": " + elemento + "<br>");  
}
```

JS

FOR EACH

0: Banana

1: Naranja

2: Manzana

3: Mango

Recorremos el vector y por cada (for each) iteración llamamos a la función **mostrar**, que escribirá en el HTML el índice y el elemento guardado en el array, incrementando el índice en 1

## .every(cb, arg)

```
var edades = [32, 33, 16, 40];  
var edades2 = [32, 33, 20, 40];  
document.write(edades.every(compruebaEdad)); //false  
document.write(edades2.every(compruebaEdad)); //true  
function compruebaEdad(edad) {  
    return edad >= 18;  
}
```

JS

EVERY  
false  
true

Para ambos vectores nos fijamos si **todos (every)** los elementos cumplen con la condición establecida en la función.

**Primer vector:** no cumple el número 16.  
**Segundo vector:** cumplen todos.



# Array | Más métodos | some y map

## .some(cb, arg)

```
var nombres = ["Juan", "Mateo", "Camilo", "Lucas"];
var nombres2 = ["Juan", "Ana", "Luisa", "Mateo", "Camilo"];
document.write(nombres.some(compruebaNombre)); //true
document.write(nombres2.some(compruebaNombre)); //false
function compruebaNombre(nombre) {
    return nombre == "Lucas";
}
```

JS

SOME  
true  
false

En el primer array encontramos algún "Lucas", pero en el segundo array no. El método **some** necesita que alguno de los valores coincida con el valor devuelto por la función **compruebaNombre**

## .map(cb, arg)

```
var numeros = [4, 9, 16, 25];
document.write(numeros.map(raizCuadrada));
document.write("<br>");
function raizCuadrada(numero) {
    return Math.sqrt(numero);
}
```

JS

MAP  
2,3,4,5

A partir del array de números, **map** creará un nuevo array aplicando la función **raizCuadrada** y colocando lo que devuelve la función en el nuevo vector.

# Array | Más métodos | filter y findIndex

## .filter(cb, arg)

```
var personas = ["Ana", "Pablo", "Pedro", "Paola", "Horacio"];
document.write(personas.filter(personasComiezaEnP));
function personasComiezaEnP(persona) {
    return persona[0] == "P";
}
```

JS

FILTER  
Pablo,Pedro,Paola

A partir del array de personas, **filter** creará un nuevo array mostrando (filtrando) solamente aquellos que cumplan con la condición de que la primer letra del cada elemento [0] sea "P", es decir, aquellos cuyo nombre comienza con P.

## .findIndex(cb, arg)

```
var edades3 = [30, 19, 10, 28];
document.write(edades3.findIndex(compruebaMenorEdad));
function compruebaMenorEdad(edad) {
    if (edad < 18) {
        return edad;
    }
}
```

JS

FIND INDEX  
2

A partir del array de edades, **findIndex** buscará la posición del valor que cumple con la condición dada en la función (que la edad sea menor a 18 años)

# Array | Más métodos | find y reduce

## .find(cb, arg)

```
var edades4 = [5, 30, 19, 10, 28];
document.write(edades4.find(compruebaMenorEdad));
function compruebaMenorEdad(edad) {
    if (edad < 18) {
        return edad;
    }
}
```

JS

FIND  
5

A partir del array de edades, **find** buscará el valor que cumple con la condición dada en la función (que la edad sea menor a 18 años)

## .reduce(cb, arg) y .reduceRight(cb, arg)

```
var precios = [110, 10, 25, 50, 15];
document.write(precios.reduce(restaPrecios));
document.write(precios.reduceRight(restaPrecios));
function restaPrecios(total, p) {
    return total - p;
}
```

JS

REDUCE y REDUCE RIGHT  
10  
-180

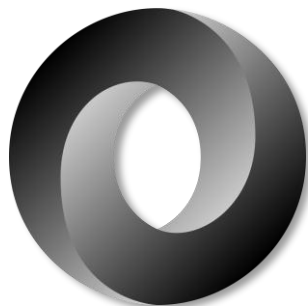
A partir del array de precios, **reduce** toma el primer elemento y acumula el resultado de izquierda a derecha, en este caso una resta desde el primer valor. Por su parte, **reduceRight** hace lo propio, pero de derecha a izquierda.

# JSON: JavaScript Object Notation

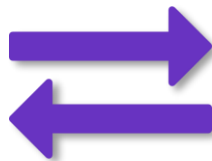
**JSON** es una sintaxis propia de objetos tipo JavaScript utilizada para **almacenar e intercambiar** datos. Es texto, escrito con notación de objetos JavaScript, con un formato determinado.

## Intercambio de datos

Al intercambiar datos entre un navegador y un servidor, los datos **solo pueden ser texto**. Dado que JSON trabaja como texto podemos convertir cualquier objeto JavaScript en JSON y enviar JSON al servidor. También podemos convertir cualquier JSON recibido del servidor en objetos JavaScript. De esta forma podemos trabajar con los datos como objetos JavaScript, sin complicados análisis ni traducciones.



**JSON**



*Cualquier JSON podrá ser  
convertido a JS y cualquier texto  
con el formato correspondiente  
podrá ser convertido a JSON*



# JSON: JavaScript Object Notation

## Convirtiendo de JavaScript a JSON:

Si tiene datos almacenados en un objeto JavaScript, puede convertir el objeto en JSON con ***JSON.stringify()***:

```
var myObj = { name: "John", age: 31, city: "New York" };  
var myJSON = JSON.stringify(myObj);  
// myJson= {"name":"John","age":31,"city":"New York"}
```

JS

Más info [aquí](#)

*[Ver ejemplo  
JstoJSON.html](#)*

## Convirtiendo de JSON a JavaScript:

Si tiene datos almacenados en un JSON, puede convertir el objeto en JavaScript con ***JSON.parse()***:

```
var myObj1=JSON.parse(myJSON);  
//myObj1= { name: "John", age: 31, city: "New York" }
```

JS

Más info [aquí](#)

*[Ver ejemplo  
JSONtoJS.html](#)*



# JSON: JavaScript Object Notation

## Reglas de sintaxis JSON:

La sintaxis JSON se deriva de la sintaxis de notación de objetos de JavaScript:

- Los datos están en pares de nombre / valor
- Los datos están separados por comas
- Las {} contienen objetos
- Los corchetes contienen Array

```
myJson= {"name":"John","age":31,"city":"New York"}
```

JS

En JSON , los valores deben ser uno de los siguientes tipos de datos:

- string
- number
- object (JSON object)
- array
- boolean
- null

El tipo de archivo de los archivos JSON es ".json"



# JSON: JavaScript Object Notation

## JSON

```
// Ejemplo de JSON:
{
  "empleados": [
    { "nombre": "Juan", "apellido": "Pérez" },
    { "nombre": "Ana", "apellido": "López" },
    { "nombre": "Pedro", "apellido": "Fernández" }
  ]
}

// Otro ejemplo:
{
  "nombre": "Carlos",
  "apellido": "Rivarola",
  "segundoNombre": null,
  "edad": 30,
  "hijos": ["Ana", "Luisa", "Marcelo"]
}
```

**1er ejemplo:** En la propiedad “empleados” hay un array de 3 elementos y dentro de cada uno tengo objetos separados por comas. Cada objeto tiene un nombre y un apellido.

**2do ejemplo:** Vemos que el objeto JSON tiene un primer nombre asociado, junto con otras propiedades. Además hay una propiedad hijos que a su vez tiene un array.

*Ver ejemplos [ejemplo.json](#) y [ejemplo2.json](#)*

# JSON: JavaScript Object Notation

Otro ejemplo JSON: Ingresar aquí: <https://mdn.github.io/learning-area/javascript/ojs/json/superheroes.json>

*Ver ejemplo  
superheroes.json*

```
{
  "squadName" : "Super Hero Squad",
  "homeTown" : "Metro City",
  "formed" : 2016,
  "secretBase" : "Super tower",
  "active" : true,
  "members" : [
    {
      "name" : "Molecule Man",
      "age" : 29,
      "secretIdentity" : "Dan Jukes",
      "powers" : [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name" : "Madame Uppercut",
      "age" : 39,
      "secretIdentity" : "Jane Wilson",
      "powers" : [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    }
  ]
}
```

Google Chrome



```
squadName:      "Super Hero Squad"
homeTown:       "Metro City"
formed:         2016
secretBase:     "Super tower"
active:         true
▼ members:
  ▼ 0:
    name:        "Molecule Man"
    age:         29
    secretIdentity: "Dan Jukes"
    ▼ powers:
      0:          "Radiation resistance"
      1:          "Turning tiny"
      2:          "Radiation blast"
  ▼ 1:
    name:        "Madame Uppercut"
    age:         39
    secretIdentity: "Jane Wilson"
    ▼ powers:
      0:          "Million tonne punch"
      1:          "Damage resistance"
      2:          "Superhuman reflexes"
```

Firefox Developer



Otro ejemplo JSON: <https://github.com/midesweb/taller-angular/blob/master/11-mi-API/peliculas.json>

*Ver ejemplo  
películas.json*



# JSON: JavaScript Object Notation

**API pública Randomuser:** <https://randomuser.me/api>

Muestra datos de usuarios aleatorios, se utiliza para hacer pruebas. Es un string de JSON con un formato particular. Devuelve un usuario aleatorio, un array con un solo elemento.

Conviene leerlo desde **Firefox Developer Edition**, ya que la visualización es más simple.

Nosotros podremos **consumir la API**, esto quiere decir leerla y traerla a nuestra aplicación. Ingresando en <https://randomuser.me/api/?results=5> podremos obtener 5 resultados, por ejemplo.

# JSON: JavaScript Object Notation

**Curso de JSON (lista de reproducción . Ver videos 1 y 2):**

<https://www.youtube.com/playlist?list=PLrAw40DbN0l0P8JZRrRUXYsFw98Q768WI>

**Página oficial de JSON:** <https://www.json.org/json-es.html>

**Cargar archivo JSON en JavaScript:** <https://www.delftstack.com/es/howto/javascript/load-json-file-in-javascript/>



# LocalStorage

Vimos que veníamos trabajando con variables en JavaScript, pero si las modificábamos perdíamos la información. Hay una forma de almacenar esa información en formato texto, en algún lugar local. Cuando se refiere a “local” se refiere al navegador del cliente, es decir quien esté navegando el sitio Web.

Tenemos dos formas de grabar:

- **A nivel local:** Cuando cierro el navegador esa información queda almacenada.
- **A nivel de sesión:** Va a durar lo que dure la sesión, hasta que cierre el navegador.

Estas dos formas nos van a permitir almacenar información importante del usuario.

Esto me puede servir en este caso: yo sé que en el navegador el usuario siempre accede a esta parte del contenido del sitio, claramente a mi usuario le interesa eso, pero voy con otro usuario y a ese otro usuario le interesará otra parte del sitio. Mi usuario tiene que cargar datos y siempre son los mismos, eso lo puedo almacenar en algún lugar, reconocer que esa persona ya me guardó información con respecto a ella en algún lugar, accedo, levanto y no se lo vuelvo a pedir porque ya la tengo. Esto es una posibilidad de almacenamiento local que nos da JavaScript.

# LocalStorage

## Definición y uso

Las propiedades **localStorage** y **sessionStorage** permiten guardar pares clave / valor en un navegador web.

El objeto **localStorage** almacena datos sin fecha de vencimiento. Los datos no se eliminarán cuando se cierre el navegador y estarán disponibles el próximo día, semana o año.

Los datos almacenados en **sessionStorage** son eliminados cuando finaliza la sesión de navegación - lo cual ocurre cuando se cierra la página.

Los datos guardados son siempre formato texto.

**Video tutorial “Uso del local storage en JavaScript”:** <https://youtu.be/hb8O0qRqiSk>

**Cookies, localStorage y sessionStorage:** <https://www.youtube.com/watch?v=DRs6zlwKZ34>

## Más información:

[https://www.w3schools.com/jsref/prop\\_win\\_localstorage.asp](https://www.w3schools.com/jsref/prop_win_localstorage.asp)

[https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\\_win\\_localstorage](https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_win_localstorage)

# LocalStorage

## Ejemplo:

```
if (typeof(Storage) !== "undefined") {  
    // Store  
    localStorage.setItem("apellido", "Perez"); // No existe, lo agrega.  
    localStorage.setItem("apellido", "Pérez"); // Existe, lo reemplaza  
    localStorage.setItem("nombre", "Juan");  
    // Retrieve  
    document.getElementById("resultado").innerHTML = localStorage.getItem  
("apellido");  
} else {  
    document.getElementById("resultado").innerHTML = "Su navegador no sop  
orta Web Storage."  
}
```

JS

*localStorage es un objeto que tiene un método asociado `setItem`.*

*Los métodos **getters** y **setters**, me permiten acceder a información del objeto, leerla y grabarla.*

*Esta información siempre se guarda en formato de texto, con el par clave / valor.*

```
<body>  
    <div id="resultado">  
    </div>  
    <script src="localStorage.js"></script>  
</body>
```

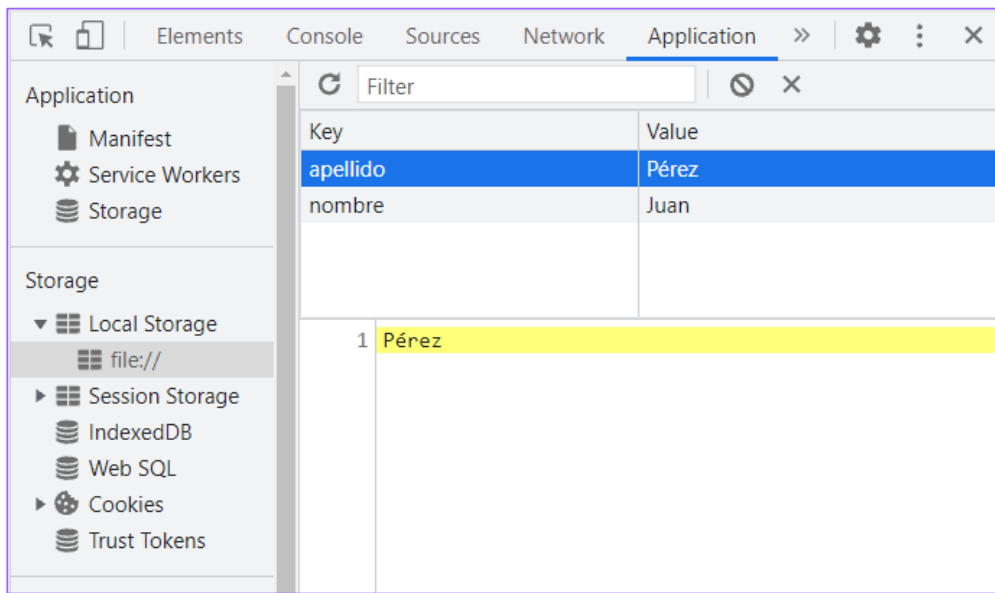
HTML

*En este caso "apellido" no existe, la estoy creando con `setItem` y le estoy dando el valor **Perez** en primera instancia y luego lo reemplazo por **Pérez**.*

# LocalStorage

## Ejemplo (continuación):

Para poder ver estos datos accedemos al navegador. Vamos a ver que en el **body** se muestra el apellido, gracias a que hemos utilizado `document.getElementById("resultado").innerHTML = localStorage.getItem("apellido");`, pero también vamos a poder verlo al inspeccionar la página yendo a la pestaña Application:



*Ver ejemplo  
localStorage  
(.html y .js)*