1. **Set time out**

It is a method that calls a function after a number on milli Seconds

setTimeout(function , 5000)

Used only once , For many uses Use SetInterval()

Use clearTimeout() for preventing the function from starting by calling the name of variable the timeout was called

**LocalStorage, sessionStorage**

Web storage objects localStorage and sessionStorage allow to save key/value pairs in the browser.

What’s interesting about them is that the data survives a page refresh (for sessionStorage) and even a full browser restart (for localStorage). We’ll see that very soon.

We already have cookies. Why additional objects?

* Unlike cookies, web storage objects are not sent to server with each request. Because of that, we can store much more. Most modern browsers allow at least 5 megabytes of data (or more) and have settings to configure that.
* Also unlike cookies, the server can’t manipulate storage objects via HTTP headers. Everything’s done in JavaScript.
* The storage is bound to the origin (domain/protocol/port triplet). That is, different protocols or subdomains infer different storage objects, they can’t access data from each other.

Both storage objects provide the same methods and properties:

* setItem(key, value) – store key/value pair.
* getItem(key) – get the value by key.
* removeItem(key) – remove the key with its value.
* clear() – delete everything.
* key(index) – get the key on a given position.
* length – the number of stored items.

As you can see, it’s like a Map collection (setItem/getItem/removeItem), but also allows access by index with key(index).

## [localStorage demo](https://javascript.info/localstorage" \l "localstorage-demo)

The main features of localStorage are:

* Shared between all tabs and windows from the same origin.
* The data does not expire. It remains after the browser restart and even OS reboot.

If the key is user-generated, it can be anything, like length or toString, or another built-in method of localStorage. In that case getItem/setItem work fine, while object-like access fails

## [Looping over keys](https://javascript.info/localstorage" \l "looping-over-keys)

One way is to loop over them as over an array

Another way is to use for key in localStorage loop, just as we do with regular objects.

It iterates over keys, but also outputs few built-in fields that we don’t need

filter fields from the prototype with hasOwnProperty check

for(let key in localStorage) {

if (!localStorage.hasOwnProperty(key)) {

continue; // skip keys like "setItem", "getItem" etc

}

alert(`${key}: ${localStorage.getItem(key)}`);

}

get the “own” keys with Object.keys and then loop over them if needed

## [Strings only](https://javascript.info/localstorage" \l "strings-only)

Please note that both key and value must be strings.

If they were any other type, like a number, or an object, they would get converted to a string automatically

We can use JSON to store objects though:

localStorage.user = JSON.stringify({name: "John"});

// sometime later

let user = JSON.parse( localStorage.user );

alert( user.name ); // John

## [sessionStorage](https://javascript.info/localstorage" \l "sessionstorage)

The sessionStorage object is used much less often than localStorage.

Properties and methods are the same, but it’s much more limited:

* The sessionStorage exists only within the current browser tab.
  + Another tab with the same page will have a different storage.
  + But it is shared between iframes in the same tab (assuming they come from the same origin).
* The data survives page refresh, but not closing/opening the tab.

## [Storage event](https://javascript.info/localstorage" \l "storage-event)

When the data gets updated in localStorage or sessionStorage, [storage](https://html.spec.whatwg.org/multipage/webstorage.html#the-storageevent-interface) event triggers, with properties:

* key – the key that was changed (null if .clear() is called).
* oldValue – the old value (null if the key is newly added).
* newValue – the new value (null if the key is removed).
* url – the url of the document where the update happened.
* storageArea – either localStorage or sessionStorage object where the update happened.

**Cookies**

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

* When a user visits a web page, his/her name can be stored in a cookie.
* Next time the user visits the page, the cookie "remembers" his/her name.
* When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

JavaScript can create, read, and delete cookies with the document.cookie property.

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

## Delete a Cookie with JavaScript

Deleting a cookie is very simple.

You don't have to specify a cookie value when you delete a cookie.

Just set the expires parameter to a past date:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";

## The Cookie String

The document.cookie property looks like a normal text string. But it is not.

Even if you write a whole cookie string to document.cookie, when you read it out again, you can only see the name-value pair of it.

If you set a new cookie, older cookies are not overwritten. The new cookie is added to document.cookie

## JavaScript Cookie Example

In the example to follow, we will create a cookie that stores the name of a visitor.

The first time a visitor arrives to the web page, he/she will be asked to fill in his/her name. The name is then stored in a cookie.

The next time the visitor arrives at the same page, he/she will get a welcome message.

For the example we will create 3 JavaScript functions:

1. A function to set a cookie value
2. A function to get a cookie value
3. A function to check a cookie value
4. First, we create a function that stores the name of the visitor in a cookie variable:
5. function setCookie(cname, cvalue, exdays) {  
     const d = new Date();  
     d.setTime(d.getTime() + (exdays\*24\*60\*60\*1000));  
     let expires = "expires="+ d.toUTCString();  
     document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";  
   }

then, we create a function that returns the value of a specified cookie:

function getCookie(cname) {  
  let name = cname + "=";  
  let decodedCookie = decodeURIComponent(document.cookie);  
  let ca = decodedCookie.split(';');  
  for(let i = 0; i <ca.length; i++) {  
    let c = ca[i];  
    while (c.charAt(0) == ' ') {  
      c = c.substring(1);  
    }  
    if (c.indexOf(name) == 0) {  
      return c.substring(name.length, c.length);  
    }  
  }  
  return "";  
}

**Function explained:**

Take the cookiename as parameter (cname).

Create a variable (name) with the text to search for (cname + "=").

Decode the cookie string, to handle cookies with special characters, e.g. '$'

Split document.cookie on semicolons into an array called ca (ca = decodedCookie.split(';')).

Loop through the ca array (i = 0; i < ca.length; i++), and read out each value c = ca[i]).

If the cookie is found (c.indexOf(name) == 0), return the value of the cookie (c.substring(name.length, c.length).

If the cookie is not found, return "".

Last, we create the function that checks if a cookie is set.

If the cookie is set it will display a greeting.

If the cookie is not set, it will display a prompt box, asking for the name of the user, and stores the username cookie for 365 days, by calling the setCookie function:

function checkCookie() {  
  let username = getCookie("username");  
  if (username != "") {  
   alert("Welcome again " + username);  
  } else {  
    username = prompt("Please enter your name:", "");  
    if (username != "" && username != null) {  
      setCookie("username", username, 365);  
    }  
  }  
}