[**Cookies**](http://il.php.net/manual/en/features.cookies.php) and [**sessions**](http://il.php.net/manual/en/features.sessions.php) are both ways to preserve the application's state between different requests the browser makes. It's thanks to them that, for instance, you don't need to log in every time you request a page on **[StackOverflow](http://stackoverflow.com/)**.

Cookies

Cookies are small bits of data, (maximum of 4KB long), which hold data in a key=value pairs:

name=value; name2=value2

These are set either by [**JavaScript**](https://developer.mozilla.org/en/DOM/document.cookie), or via the server using an [**HTTP header**](http://il.php.net/manual/en/function.setcookie.php).

Cookies have an expiry datetime set, example using HTTP headers:

Set-Cookie: name2=value2; Expires=Wed, 19 Jun 2021 10:18:14 GMT

Which would cause the browser to set a cookie named name2 with a value of value2, which would expire in about 9 years.

Cookies are considered **highly insecure** because the user can easily manipulate their content. That's why you should **always validate cookie data**. Don't assume what you get from a cookie is necessarily what you expect.

Cookies are usually used to preserve login state, where a username and a special hash are sent from the browser, and the server checks them against the database to approve access.

Cookies are also often used in **sessions** creation.

Sessions

Sessions are slightly different. Each user gets a **session ID**, which is sent back to the server for validation either by **cookie** or by **GET variable**.

Sessions are usually short-lived, which makes them ideal in saving temporary state between applications. Sessions also expire once the user closes the browser.

Sessions are considered more secure than cookies because the variables themselves are kept on the **server**. Here's how it works:

1. Server opens a session (sets a cookie via HTTP header)
2. Server sets a session variable.
3. Client changes page
4. Client sends all cookies, along with the session ID from step 1.
5. Server reads session ID from cookie.
6. Server matches session ID from a list in a database (or memory etc).
7. Server finds a match, reads variables which are now available on $\_SESSION superglobal.

If PHP does not find a match, it will start a new session, and repeat the steps from 1-7.

You can store sensitive information on a session because it is kept on the server, but be aware that the session ID can still be stolen if the user, let's say, logged in over an insecure WiFi. (An attacker can sniff the cookies, and set it as its own, he won't see the variables themselves, but the server will identify the attacker as the user).

1. 2011

The concept is storing persistent data across page loads for a web visitor. Cookies store it directly on the client. Sessions use a cookie as a key of sorts, to associate with the data that is stored on the server side.

It is preferred to use sessions because the actual values are hidden from the client, and you control when the data expires and becomes invalid. If it was all based on cookies, a user (or hacker) could manipulate their cookie data and then play requests to your site.

Edit: I don't think there is any advantage to using cookies, other than simplicity. Look at it this way... Does the user have any reason to know their ID#? Typically I would say no, the user has no need for this information. Giving out information should be limited on a need to know basis. What if the user changes his cookie to have a different ID, how will your application respond? It's a security risk.

Before sessions were all the rage, I basically had my own implementation. I stored a unique cookie value on the client, and stored my persistent data in the database along with that cookie value. Then on page requests I matched up those values and had my persistent data without letting the client control what that was.

Pretty sure that typically the session ID gets stored AS a cookie on the client machine, and is then matched up server-side with the session data. The server does not typically control sessions via IP address, rather through a cookie value.

Basic ideas to distinguish between those two.

**Session:**

1. IDU is stored on server (i.e. server-side)
2. Safer (because of 1)
3. Expiration can not be set, session variables will be expired when users close the browser. (nowadays it is stored for 24 minutes as default in php)

**Cookies:**

1. IDU is stored on web-browser (i.e. client-side)
2. Not very safe, since hackers can reach and get your information (because of 1)
3. Expiration can be set (see [setcookies()](http://php.net/manual/en/function.setcookie.php) for more information)

Session is preferred when you need to store short-term information/values, such as variables for calculating, measuring, querying etc.

Cookies is preferred when you need to store long-term information/values, such as user's account (so that even when they shutdown the computer for 2 days, their account will still be logged in). I can't think of many examples for cookies since it isn't adopted in most of the situations.

SESSIONS ENDS WHEN USER CLOSES THEIR BROWSER,

COOKIES END DEPENDING ON THE LIFETIME YOU SET FOR IT. SO THEY CAN LAST FOR YEARS

This is the major difference in your choice,

If you want the id to be remembered for long time, then you need to use cookies; otherwise if you just want the website to recognize the user for this visit only then sessions is the way to go.

Sessions are stored in a file your php server will generate. To remember which file is for which user, php will also set a cookie on the user's browser that holds this session file id so in their next visit php will read this file and reload the session.

Sessions are server-side files that contain user information, while Cookies are client-side files that contain user information. Sessions have a unique identifier that maps them to specific users. This identifier can be passed in the URL or saved into a session cookie.

Most modern sites use the second approach, saving the identifier in a Cookie instead of passing it in a URL (which poses a security risk). You are probably using this approach without knowing it, and by deleting the cookies you effectively erase their matching sessions as you remove the unique session identifier contained in the cookies

A cookie is simply a short text string that is sent back and forth between the client and the server. You could store name=bob; password=asdfas in a cookie and send that back and forth to identify the client on the server side. You could think of this as carrying on an exchange with a bank teller who has no short term memory, and needs you to identify yourself for each and every transaction. Of course using a cookie to store this kind information is horrible insecure. Cookies are also limited in size.

Now, when the bank teller knows about his/her memory problem, He/She can write down your information on a piece of paper and assign you a short id number. Then, instead of giving your account number and driver's license for each transaction, you can just say "I'm client 12"

Translating that to Web Servers: The server will store the pertinent information in the session object, and create a session ID which it will send back to the client in a cookie. When the client sends back the cookie, the server can simply look up the session object using the ID. So, if you delete the cookie, the session will be lost.

One other alternative is for the server to use URL rewriting to exchange the session id.

Suppose you had a link - www.myserver.com/myApp.jsp You could go through the page and rewrite every URL as www.myserver.com/myApp.jsp?sessionID=asdf or even www.myserver.com/asdf/myApp.jsp and exchange the identifier that way. This technique is handled by the web application container and is usually turned on by setting the configuration to use cookieless sessions.

Cookies and session both store information about the user (to make the HTTP request stateful) but the difference is that cookies store information on the client-side (browser) and sessions store information on the server-side. A cookie is limited in the sense that it stores information about limited users and only stores limited content for each user. A session is not limit in such a way.

1. 2019

Cookies

* Cookies are stored on the client side (in the visitor's browser).
* Cookies are not safe: it's quite easy to read and write cookie contents.
* When using cookies, you have to notify visitors according to european laws (GDPR).
* Expiration can be set, but user or browser can change it.
* Users (or browser) can (be set to) decline the use of cookies.

Sessions

* Sessions are stored on the server side.
* Sessions use cookies (see below).
* Sessions are safer than cookies, but not invulnarable.
* Expiration is set in server configuration (php.ini for example).
* Default expiration time is 24 minutes or when the browser is closed.
* Expiration is reset when the user refreshes or loads a new page.
* Users (or browser) can (be set to) decline the use of cookies, therefore sessions.
* Legally, you also have to notify visitors for the cookie, but the lack of precedent is not clear yet.

The appropriate choice

**Sessions use a cookie!** Session data is stored on the server side, but a UID is stored on client side in a cookie. It allows the server to match a given user with the right session data. UID is protected and hard to hack, but not invulnarable. For sensitive actions (changing email or resetting password), do not rely on sessions neither cookies : ask for the user password to confirm the action.

**Sensitive data** should never be stored in cookies (emails, encrypted passwords, personal data ...). Keep in mind the data are stored on a foreign computer, and if the computer is not private (classroom or public computers) someone else can potentially read the cookies content.

**Remember-me** data must be stored in cookies, otherwise data will be lost when the user closes the browser. However, don't save password or user personal data in the 'remember-me' cookie. Store user data in database and link this data with an encrypted pair of ID / key stored in a cookie.

After considering the previous recommandations, the following question is finally what helps you choosing between cookies and sessions:

**Must persistent data remain when the user closes the browser ?**

* If the answer is **yes**, use **cookies**.
* If the answer is **no**, use **sessions**.

Stateless applications

* Web application servers are generally "stateless":
  + Each HTTP request is independent; server can't tell if 2 requests came from the same browser or user.
  + Web server applications maintain no information in memory from request to request (only information on disk survives from one request to another).
* Statelessness not always convenient for application developers: need to tie together a series of requests from the same user.

Browser cookies

* Cookie basics:
  + The first time a browser connects with a particular server, there are no cookies.
  + When the server responds it includes a Set-Cookie: header that defines a cookie.
  + Each cookie is just a name-value pair.
  + In the future whenever the browser connects with the same server, it includes a Cookie: header containing the name and value, which the server can use to connect related requests.
* What's in a cookie?
  + Name and data.
    - Data size limited by browsers (typically < 4 KB).
    - A server can define multiple cookies with different names, but browsers limit the number of cookies per server (around 50).
  + Domain for this cookie: server, port (optional), URL prefix (optional). The cookie is only included in requests matching its domain.
  + Expiration date: browser can delete old cookies.

Sessions

* Cookies are used by the server to implement *sessions*:
  + A pool of data related to an active connection (one browser instance).
* Typically the cookie for an application contains an identifier for a session.
* Web frameworks like Rails do most of the work of managing sessions and cookies:
  + Rails provides session, a hash-like object in which you can store anything you like
    - Data will be available in all future requests from the same browser.
  + Rails automatically checks for a session cookie at the start of each request:
    - Cookie exists? use it to find session data
    - No cookie? Create new session, new cookie
  + End of each request: save session data where it can be found by future requests.
* Managing session state:
  + Approach #1: just keep state in main memory
  + Approach #2: store session state in files on disk
  + Approach #3: store session state in a database
  + Most frameworks allow you to control session storage:
    - Provide an object that saves and restores session data.