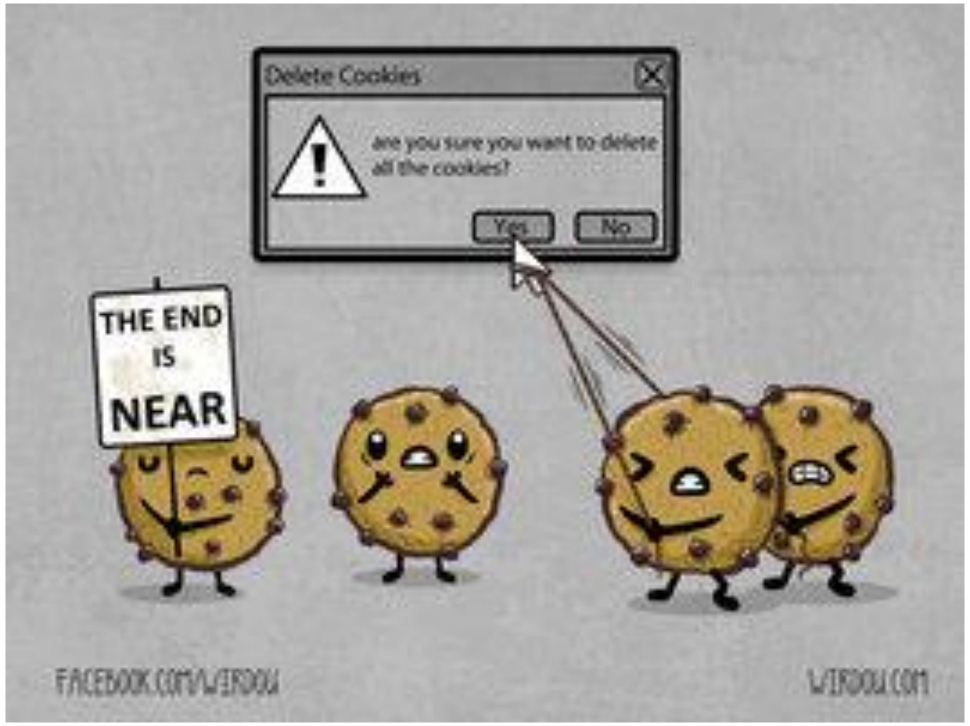
CSE 154



LECTURE 21: SESSIONS

Expiration / persistent cookies

```php
setcookie("name", "value", expiration);                                PHP
```

```php
$expireTime = time() + 60*60*24*7;    # 1 week from now
setcookie("CouponNumber", "389752", $expireTime);
setcookie("CouponValue", "100.00", $expireTime);                       PHP
```

- to set a persistent cookie, pass a third parameter for when it should expire
- indicated as an integer representing a number of seconds, often relative to current timestamp
- if no expiration passed, cookie is a session cookie; expires when browser is closed
- `time` function returns the current time in seconds
    - `date` function can convert a time in seconds to a readable date

Deleting a cookie

```
setcookie("name", FALSE);                                          PHP
```

```
setcookie("CouponNumber", FALSE);                                  PHP
```

- setting the cookie to FALSE erases it

- you can also set the cookie but with an expiration that is before the present time:

```
setcookie("count", 42, time() - 1);                      PHP
```

- remember that the cookie will also be deleted automatically when it expires, or can be deleted manually by the user by clearing their browser cookies

What is a session?

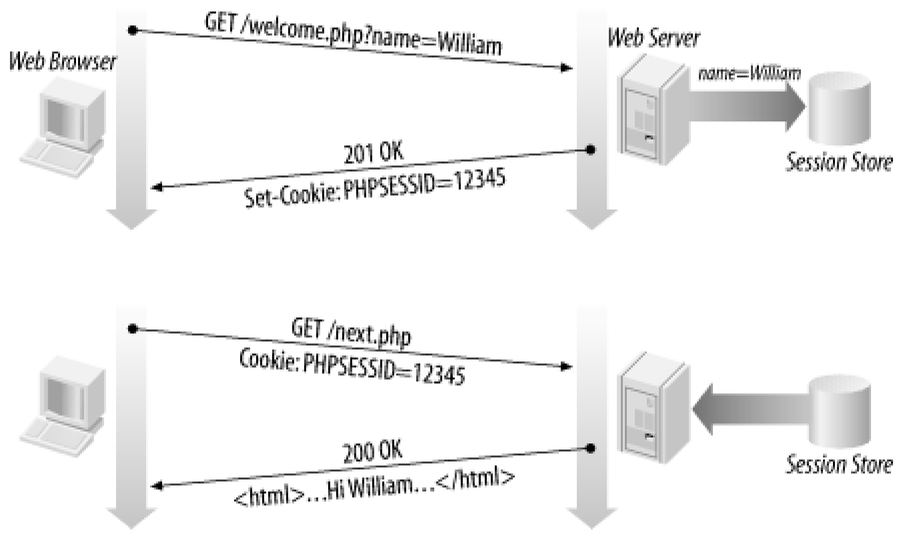
- **session**: an abstract concept to represent a series of HTTP requests and responses between a specific Web browser and server
  - HTTP doesn't support the notion of a session, but PHP does

- sessions vs. cookies:
  - a cookie is data stored on the client
  - a session's data is stored on the server (only 1 session per client)

- sessions are often built on top of cookies:
  - the only data the client stores is a cookie holding a unique **session ID**
  - on each page request, the client sends its session ID cookie, and the server uses this to find and retrieve the client's session data

How sessions are established

- client's browser makes an initial request to the server
- server notes client's IP address/browser, stores some local session data, and sends a **session ID** back to client (as a cookie)
- client sends that same session ID (cookie) back to server on future requests
- server uses session ID cookie to retrieve its data for the client's session later (like a ticket given at a coat-check room)

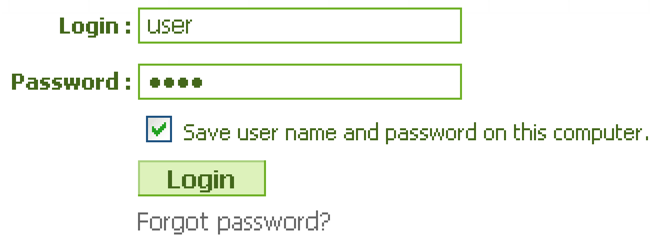


Cookies vs. sessions

- **duration:** sessions live on until the user logs out or closes the browser; cookies can live that long, or until a given fixed timeout (persistent)
- **data storage location:** sessions store data on the server (other than a session ID cookie); cookies store data on the user's browser
- **security:** sessions are hard for malicious users to tamper with or remove; cookies are easy
- **privacy:** sessions protect private information from being seen by other users of your computer; cookies do not

Implementing user logins

- many sites have the ability to create accounts and log in users
- most apps have a database of user accounts
- when you try to log in, your name/pw are compared to those in the database



Login : user
Password : ●●●●
✓ Save user name and password on this computer.

Sessions in PHP: session_start

```
session_start();                                                    PHP
```

- `session_start` signifies your script wants a session with the user
  - must be called at the top of your script, before any HTML output is produced
- when you call `session_start`:
  - if the server hasn't seen this user before, a new session is created
  - otherwise, existing session data is loaded into `$_SESSION` associative array
  - you can store data in `$_SESSION` and retrieve it on future pages
- complete list of PHP session functions

Accessing session data

```
$_SESSION["name"] = value;          # store session data
$variable = $_SESSION["name"];      # read session data
if (isset($_SESSION["name"])) {     # check for session data   PHP
```

```
if (isset($_SESSION["points"])) {
  $points = $_SESSION["points"];
  print("You've earned $points points.\n");
} else {
  $_SESSION["points"] = 0;  # default
}                                                            PHP
```

- the $_SESSION associative array reads/stores all session data
- use isset function to see whether a given value is in the session

Common session bugs

- `session_start` doesn't just begin a session; it also reloads any existing session for this user. So it must be called in every page that uses your session data:

```
# the user has a session from a previous page
print $_SESSION["name"];    # undefined

session_start();
print $_SESSION["name"];    # joe                                   PHP
```

- previous sessions will linger unless you destroy them and regenerate the user's session ID:

```
session_destroy();
session_regenerate_id(TRUE);
session_start();                                                     PHP
```

Ending a session

```
session_destroy();                                                PHP
```

- `session_destroy` ends your current session
- potential problem: if you call `session_start` again later, it sometimes reuses the same session ID/data you used before
- if you may want to start a completely new empty session later, it is best to flush out the old one:

```
session_destroy();
session_regenerate_id(TRUE);    # flushes out session
                                #ID number
session_start();                                                  PHP
```

Session timeout

- because HTTP is stateless, it is hard for the server to know when a user has finished a session
- ideally, user explicitly logs out, but many users don't
- client deletes session cookies when browser closes
- server automatically cleans up old sessions after a period of time
    - old session data consumes resources and may present a security risk
    - adjustable in PHP server settings or with `session_cache_expire` function
    - you can explicitly delete a session by calling `session_destroy`