Remembering Users: Cookies and Sessions

Chapter 10

Intro

- A web server (PHP Engine) receives many requests from many clients
- PHP Engine needs to memorize requests from the same client. Cookies are for this purpose.
- A **cookie** identifies a particular web client to the web server and PHP engine.
- Each time a web client makes a request, it sends the cookie along with the request.

- Every cookie has 2 parts: name and value

- Operations on cookies:
  - Setting
  - Reading
  - Deleting

Setting a Cookie

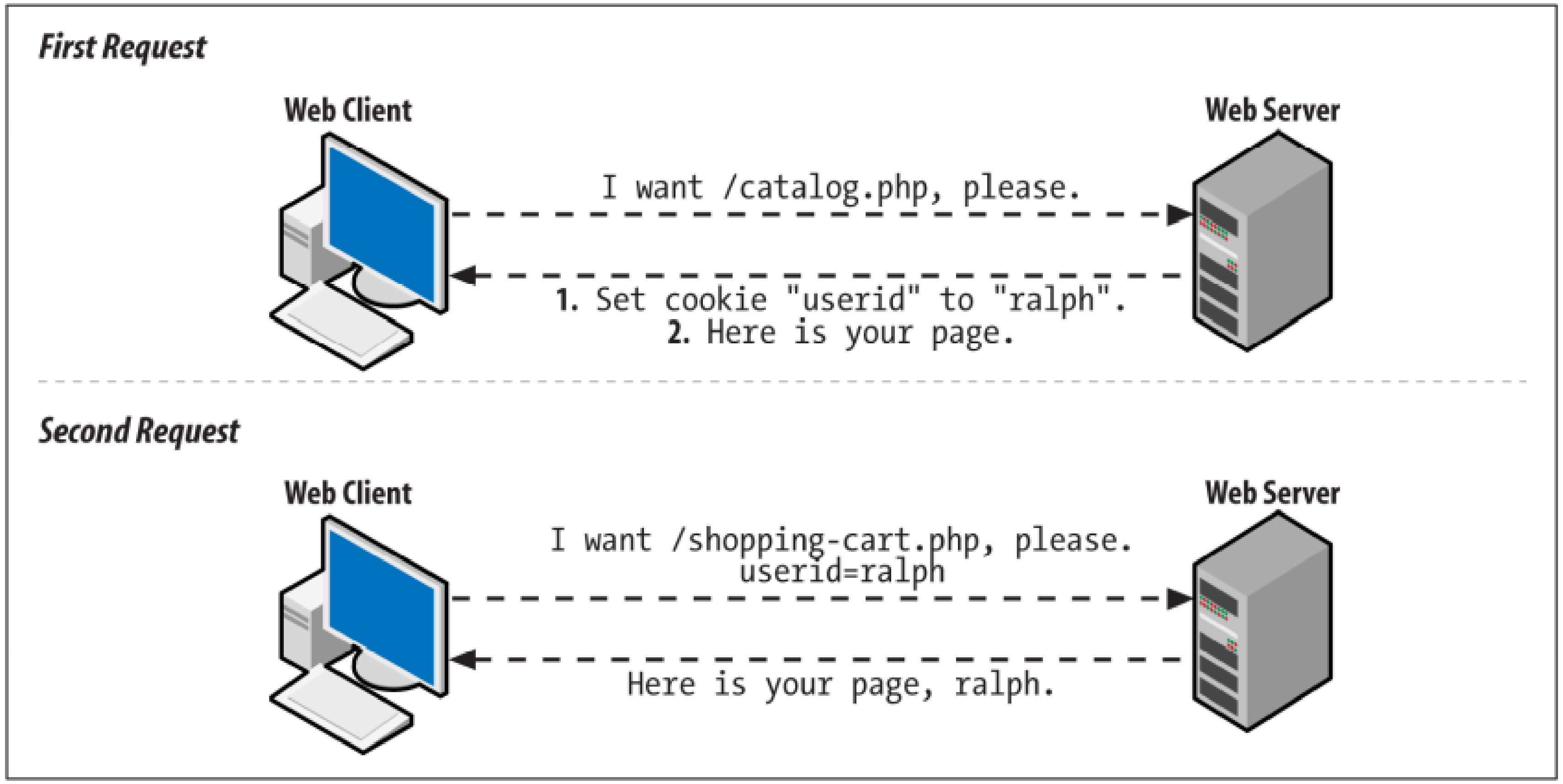

First Request
Web Client
Web Server
I want /catalog.php, please.
1. Set cookie "userid" to "ralph".
2. Here is your page.
Second Request
Web Client
Web Server
I want /shopping-cart.php, please.
userid=ralph
Here is your page, ralph.

Figure 10-1. Client and server communication when setting a cookie

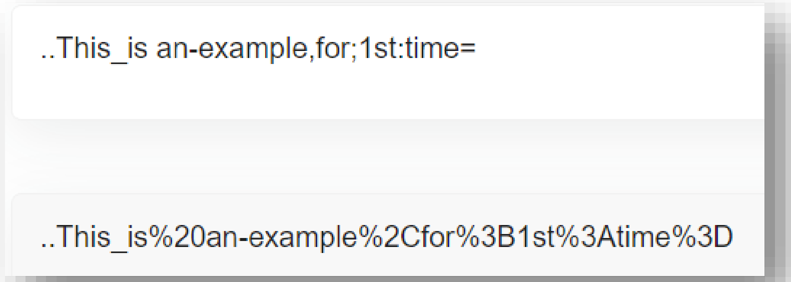
Working with Cookies – Set Cookie

- To set a cookie, use the setcookie() function

```
setcookie('userid','ralph');
```

- The value for a cookie (e.g., ralph) can be a string or a number. It can’t be an array or more complicated data structure.
- The setcookie() function URL-encodes the cookie value before sending it to the web client.
  - space is turned into a +,
  - anything other than letters, digits, underscores, hyphens, and periods is turned into a percent sign followed by its ASCII value in hexadecimal.
- Use setrawcookie() instead, If you don’t want PHP to monkey with

your cookie value



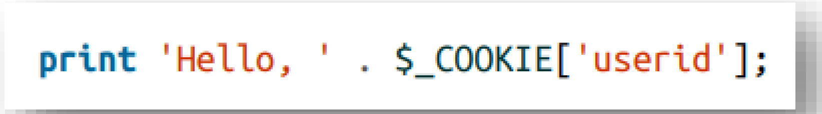
```
..This_is an-example,for;1st:time=
```

```
..This_is%20an-example%2Cfor%3B1st%3Atime%3D
```

However, with setrawcookie(), your cookie value cannot contain = , ; or any whitespace.

Working with Cookies – Read Cookie

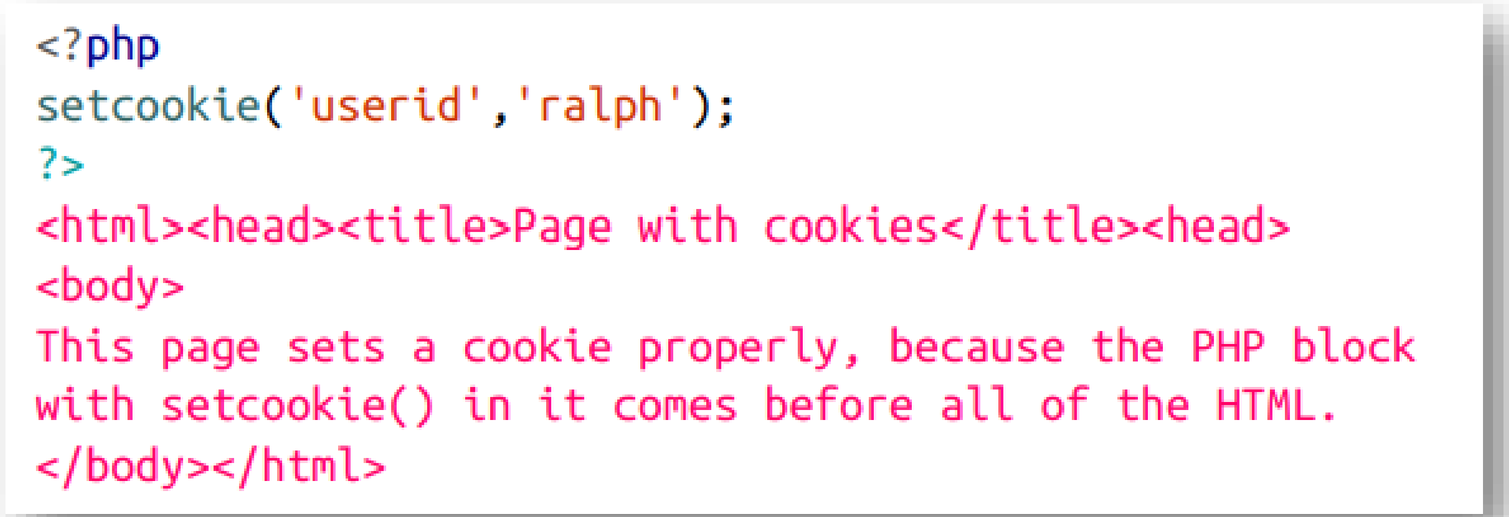
- To read a previously set cookie from your PHP program, use the $_COOKIE auto-global array.



```
print 'Hello, ' . $_COOKIE['userid'];
```

Where to Set Cookie

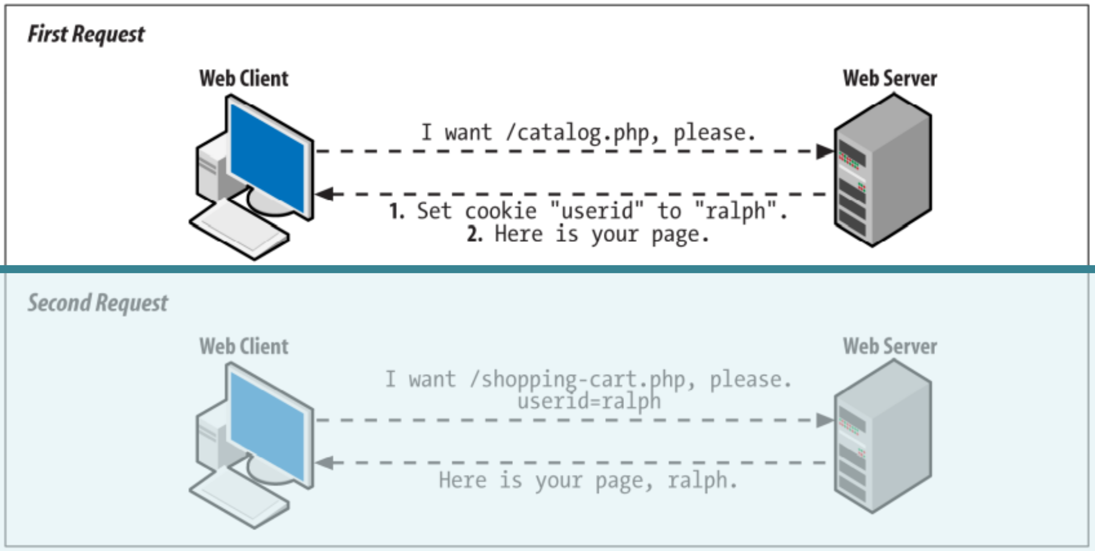
- Usually, you must call setcookie() before the page generates any output
- This means that setcookie() must come before any print statements. It also means that there can't be any text before the <?php start tag



```
<?php
setcookie('userid','ralph');
?>
<html><head><title>Page with cookies</title><head>
<body>
This page sets a cookie properly, because the PHP block
with setcookie() in it comes before all of the HTML.
</body></html>
```

Setting Cookie Expiration

- Values show up in $_COOKIE, only when the web client sends them along with the request.
- They do not appear in $_COOKIE immediately after you call setcookie().
- Only after the client sends the cookie back on a subsequent request it appear in $_COOKIE.



First Request
Web Client
Web Server
I want /catalog.php, please.
1. Set cookie "userid" to "ralph".
2. Here is your page.
Second Request
Web Client
Web Server
I want /shopping-cart.php, please.
userid=ralph
Here is your page, ralph.

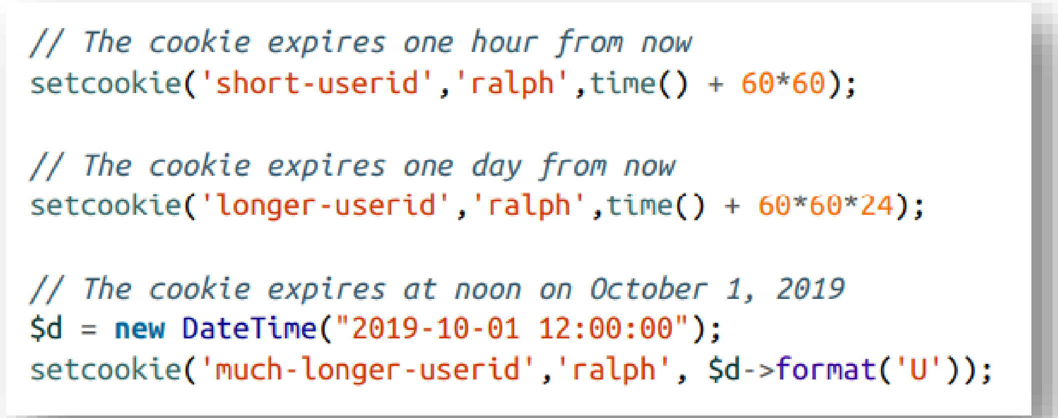
Figure 10-1. Client and server communication when setting a cookie

Setting Cookie Expiration

- The default lifetime for a cookie is the lifetime of the web client.
- When you quit Safari or Firefox, the cookie is deleted.
- To make a cookie live longer (or expire sooner), use the third argument (optional) to setcookie() in seconds.
- The value is the number of seconds elapsed since midnight on January 1, 1970 (Unix Epoch)
- Example:

1636708384 = Friday, November 12, 2021 9:13:04 AM

Learn more: Epoch Converter - Unix Timestamp Converter



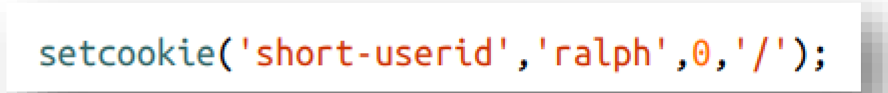
```
// The cookie expires one hour from now
setcookie('short-userid','ralph',time() + 60*60);

// The cookie expires one day from now
setcookie('longer-userid','ralph',time() + 60*60*24);

// The cookie expires at noon on October 1, 2019
$d = new DateTime("2019-10-01 12:00:00");
setcookie('much-longer-userid','ralph', $d->format('U'));
```

- 0 means the default expiration time
- time() tells you the current time in elapsed seconds since January 1, 1970 (the Unix “epoch”).
- DateTime::format(‘U’) tells you the “elapsed seconds” for the time returned by a DateTime object.

Other Cookie Setting Parameters: **Path**

- Normally, cookies are only sent back with requests for pages in the same directory (or below) as the page that set the cookie.
- A cookie set by **http://www.example.com/buy.php** is sent back with all requests to the server **www.example.com**, because buy.php is in the top-level directory of the web server.
- A cookie set by **http://www.example.com/catalog/list.php** is sent back with other requests in the catalog directory, such as **http://www.example.com/catalog/search.php**.
  http://www.example.com/catalog/detailed/search.php.
  - But not for pages above or outside the catalog directory, such as **http://www.example.com/sell.php** or http://www.example.com/users/profile.php.

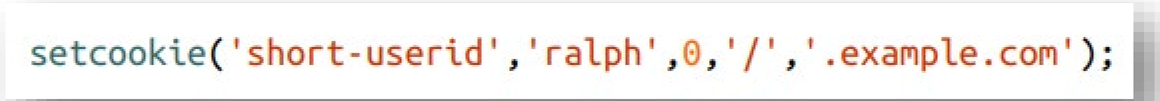
- The most flexible path is **/**, which means "send this cookie back with all requests to the server."



```
setcookie('short-userid','ralph',0,'/');
```

Other Cookie Setting Parameters: **Domain**

- The default behavior is to send cookies only with requests to the same host that set the cookie.
- Example: If http://www.example.com/login.php set a cookie,
  - then that cookie is sent back with other requests to the server www.example.com
  - but not with requests to shop.example.com, www.yahoo.com, or www.example.org.

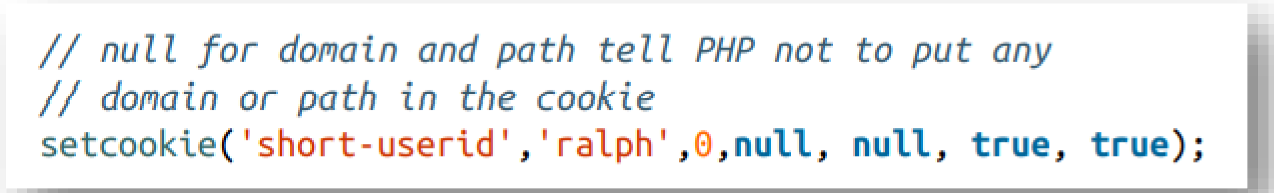


```
setcookie('short-userid','ralph',0,'/','.example.com');
```

- The path must match the end of the domain.
- Example: If your PHP programs are hosted on the server students.example.edu
  - example.edu domain
  - Not yahoo.com domain

Other Cookie Setting Parameters: **Security-related**

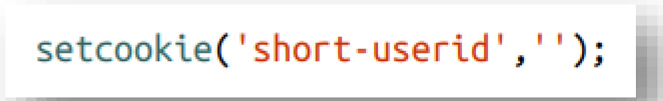
- A value of true for the 6th argument to setcookie(): client must return the cookie over a secure connection(https)
- a value of true for the 7th argument to setcookie(): the cookie is an HttpOnly cookie.
  - An HttpOnly cookie gets sent back and forth between client and server as usual, but not accessible by client-side JavaScript.
  - To prevent cross-site scripting attack



```php
// null for domain and path tell PHP not to put any
// domain or path in the cookie
setcookie('short-userid','ralph',0,null, null, true, true);
```

Delete a Cookie

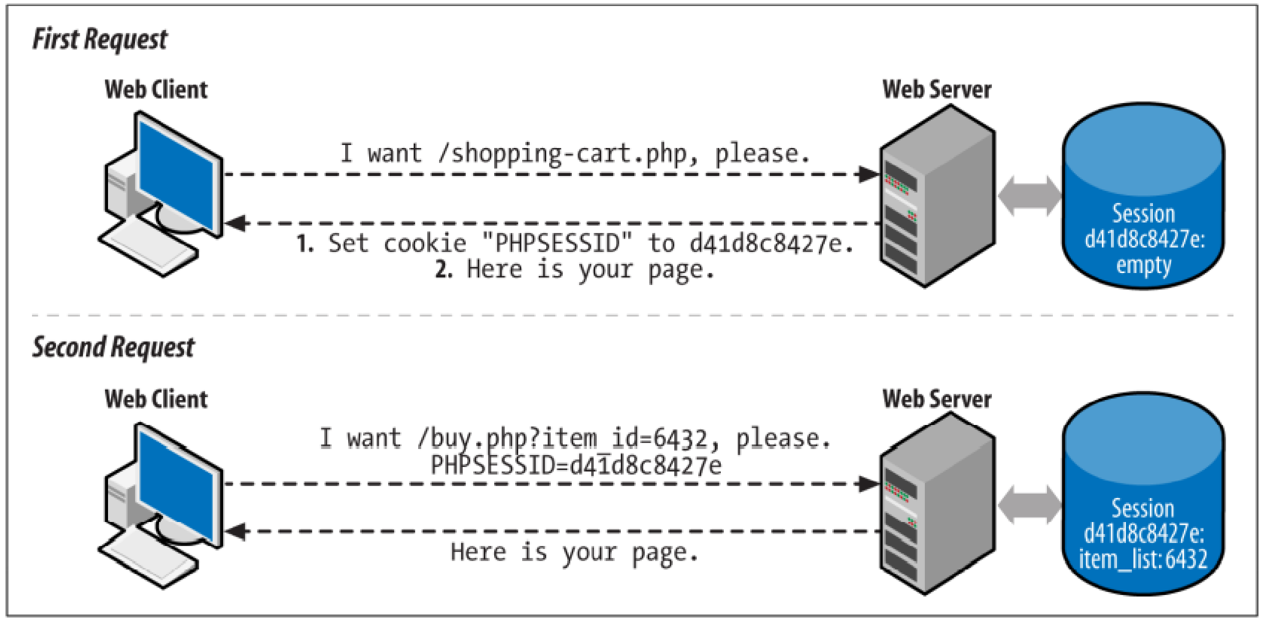
- To delete a cookie, call setcookie() with the name of the cookie, and the empty string as the cookie value



```
setcookie('short-userid','');
```

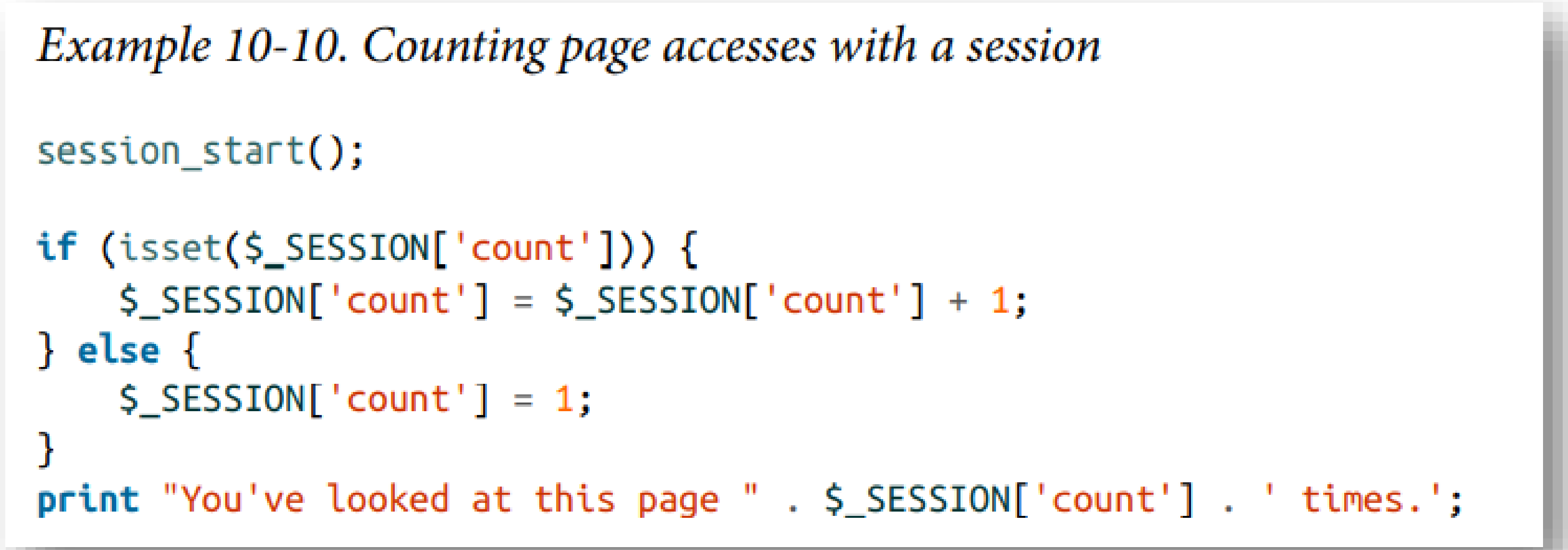
Activating Sessions

- Sessions, by default, use a cookie called PHPSESSID
- the PHP engine checks for this cookie and sets it if it doesn’t exist, When you start a session on a page.
- The value of the PHPSESSID cookie is a random alphanumeric string.


First Request
Web Client
Web Server
I want /shopping-cart.php, please.
1. Set cookie "PHPSESSID" to d41d8c8427e.
2. Here is your page.
Session d41d8c8427e: empty
Second Request
Web Client
Web Server
I want /buy.php?item_id=6432, please.
PHPSESSID=d41d8c8427e
Here is your page.
Session d41d8c8427e: item_list:6432
Figure 10-2. Client and server communication when starting a session

Use Sessions

- To use a session in a page, call session_start() at the beginning of your script.
- Like setcookie(), this function must be called before any output is sent.
- Session data is stored in the $_SESSION auto-global array.
- The $_SESSION array is a way of sharing information between pages

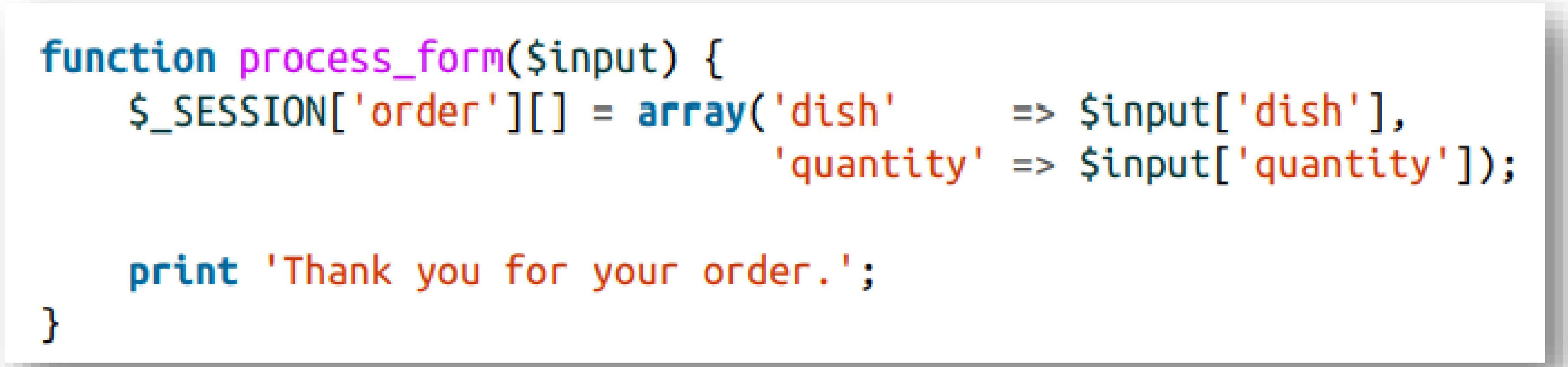


Example 10-10. Counting page accesses with a session

```
session_start();

if (isset($_SESSION['count'])) {
    $_SESSION['count'] = $_SESSION['count'] + 1;
} else {
    $_SESSION['count'] = 1;
}
print "You've looked at this page " . $_SESSION['count'] . ' times.';
```

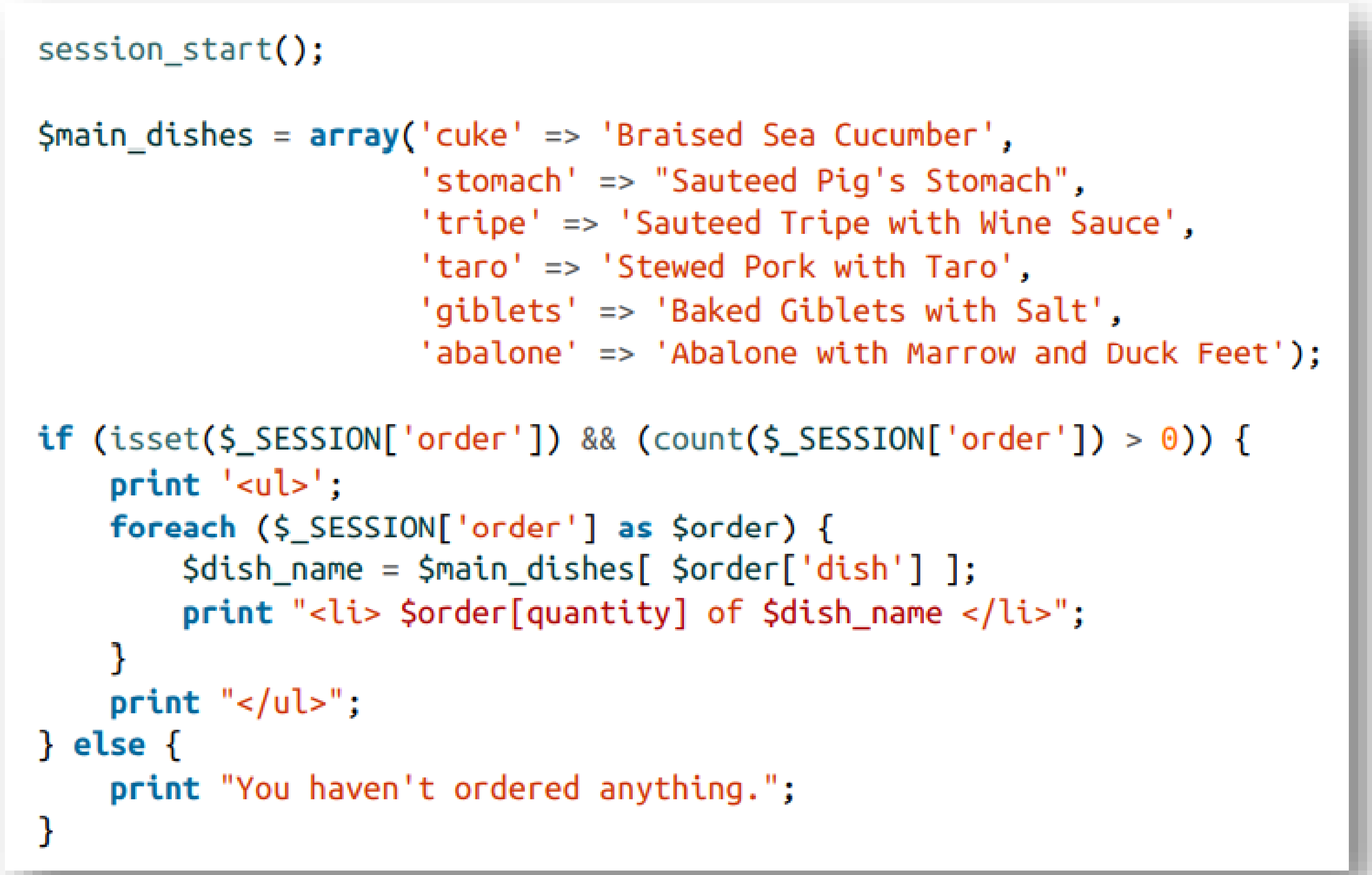
Use Sessions

- The $_SESSION array is a way of sharing information between pages



```
function process_form($input) {
    $_SESSION['order'][] = array('dish'     => $input['dish'],
                                 'quantity' => $input['quantity']);

    print 'Thank you for your order.';
}
```

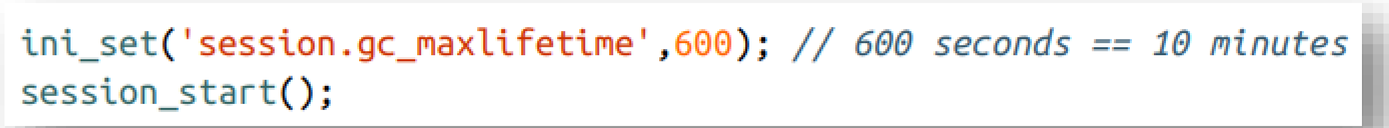
Use Sessions



```php
session_start();

$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' => 'Abalone with Marrow and Duck Feet');

if (isset($_SESSION['order']) && (count($_SESSION['order']) > 0)) {
    print '<ul>';
    foreach ($_SESSION['order'] as $order) {
        $dish_name = $main_dishes[ $order['dish'] ];
        print "<li> $order[quantity] of $dish_name </li>";
    }
    print "</ul>";
} else {
    print "You haven't ordered anything.";
}
```

Use Sessions

- Session data sticks around if the session is accessed at least once every 24 minutes.
- Sessions aren’t for a permanent data store—the database is for.
- Sessions keep track of recent user activity to make the browsing experience smoother.
- You can decrease or increase the default 24-minute.
- You can change the value of session.gc_maxlifetime in your server configuration file or by calling the ini_set() function
- If you use ini_set(), you must call it before session_start().



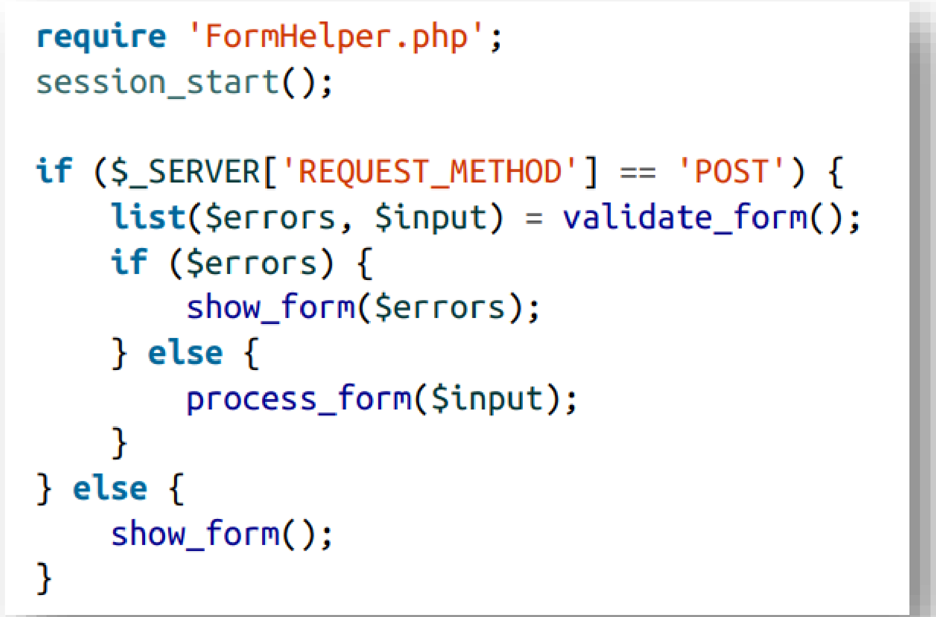
```php
ini_set('session.gc_maxlifetime',600); // 600 seconds == 10 minutes
session_start();
```

Login and User Identification

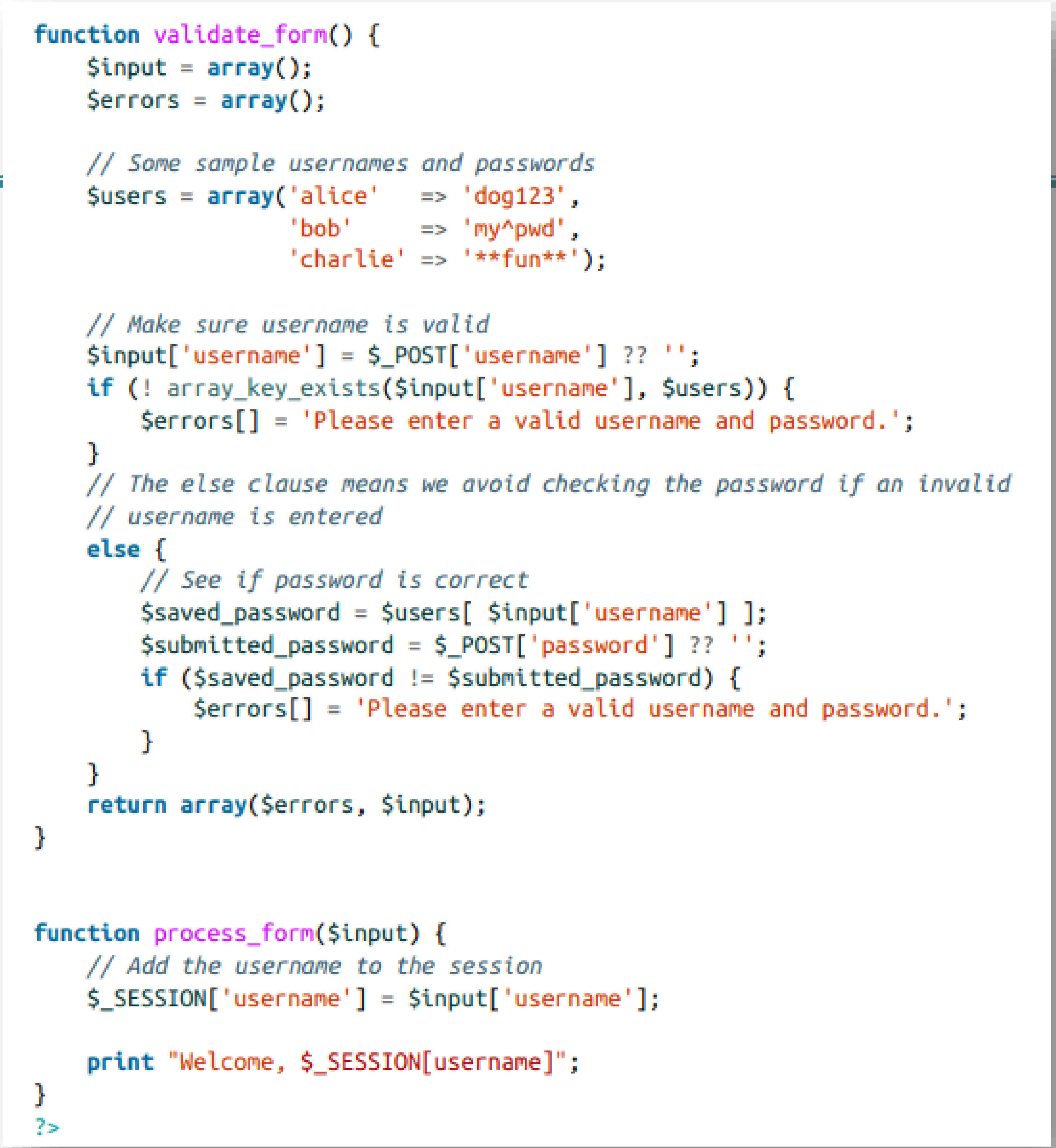
- Adding user login on top of sessions has five parts:

1. Displaying a form asking for a username and password
2. Checking the form submission
3. Adding the username to the session (if the submitted password is correct)
4. Looking for the username in the session to do user-specific tasks
5. Removing the username from the session when the user logs out

Login and User Identification



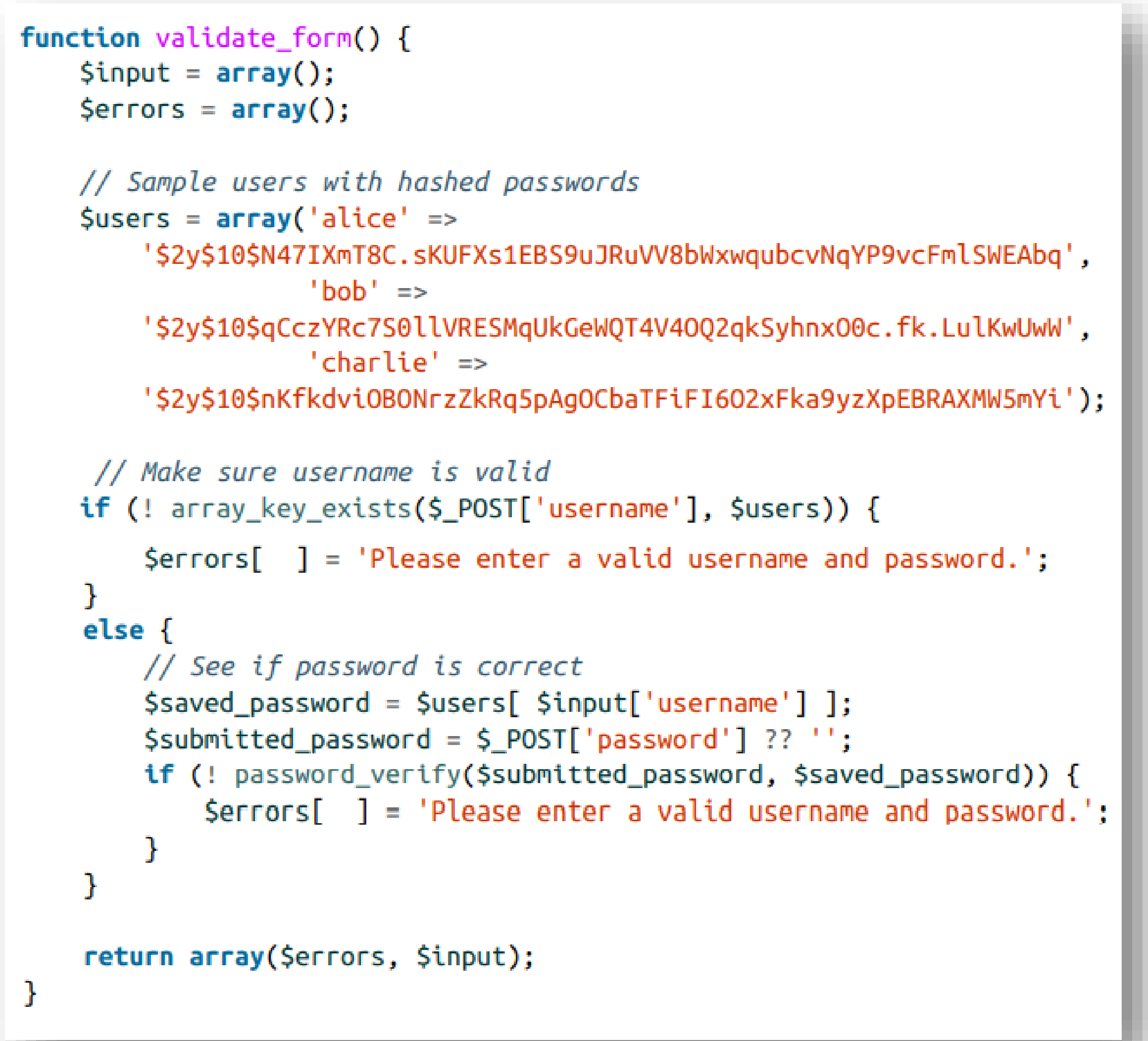
```php
require 'FormHelper.php';
session_start();

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        process_form($input);
    }
} else {
    show_form();
}
```



```php
function validate_form() {
    $input = array();
    $errors = array();

    // Some sample usernames and passwords
    $users = array('alice'   => 'dog123',
                   'bob'     => 'my^pwd',
                   'charlie' => '**fun**');

    // Make sure username is valid
    $input['username'] = $_POST['username'] ?? '';
    if (! array_key_exists($input['username'], $users)) {
        $errors[] = 'Please enter a valid username and password.';
    }
    // The else clause means we avoid checking the password if an invalid
    // username is entered
    else {
        // See if password is correct
        $saved_password = $users[ $input['username'] ];
        $submitted_password = $_POST['password'] ?? '';
        if ($saved_password != $submitted_password) {
            $errors[] = 'Please enter a valid username and password.';
        }
    }
    return array($errors, $input);
}


function process_form($input) {
    // Add the username to the session
    $_SESSION['username'] = $input['username'];

    print "Welcome, $_SESSION[username]";
}
?>
```

Login and User Identification

- Storing passwords without hashing them is a bad idea.
- If the list of unhashed passwords is compromised, then an attacker can log in as any user.
- Storing hashed passwords prevents an attacker from getting the actual passwords even if she gets the list of hashed passwords.
- Usually usernames and passwords are stored in a database table.

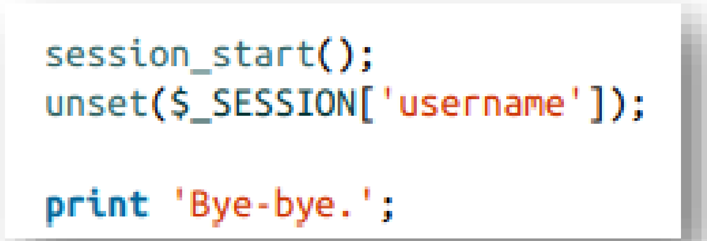


```
function validate_form() {
    $input = array();
    $errors = array();

    // Sample users with hashed passwords
    $users = array('alice' =>
        '$2y$10$N47IXmT8C.sKUFXs1EBS9uJRuVV8bWxwqubcvNqYP9vcFmlSWEAbq',
                   'bob' =>
        '$2y$10$qCczYRc7S0llVRESMqUkGeWQT4V4OQ2qkSyhnxO0c.fk.LulKwUwW',
                   'charlie' =>
        '$2y$10$nKfkdviOBONrzZkRq5pAgOCbaTFiFI6O2xFka9yzXpEBRAXMW5mYi');

     // Make sure username is valid
    if (! array_key_exists($_POST['username'], $users)) {

        $errors[  ] = 'Please enter a valid username and password.';
    }
    else {
        // See if password is correct
        $saved_password = $users[ $input['username'] ];
        $submitted_password = $_POST['password'] ?? '';
        if (! password_verify($submitted_password, $saved_password)) {
            $errors[  ] = 'Please enter a valid username and password.';
        }
    }

    return array($errors, $input);
}
```

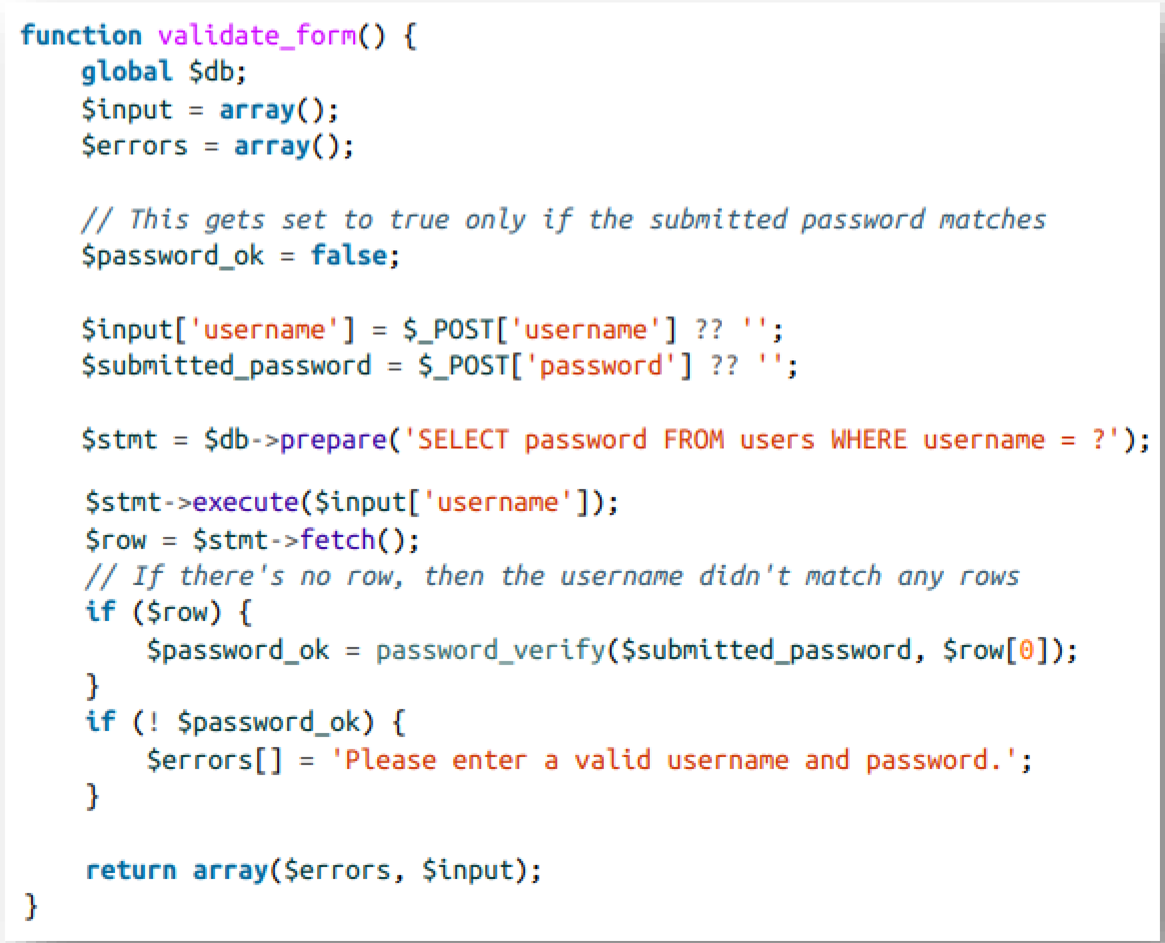
Login and User Identification

- Usually usernames and passwords are stored in a database table.

- use unset() to remove a key and value from $_SESSION.



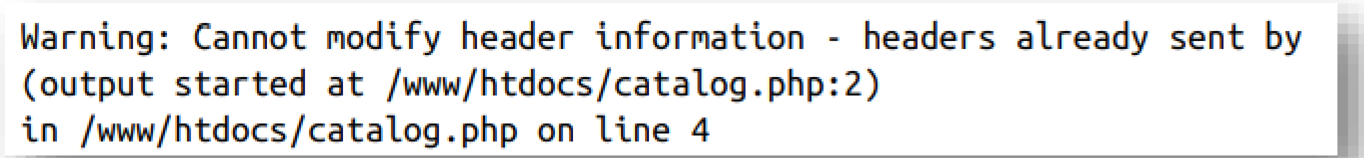
```php
session_start();
unset($_SESSION['username']);

print 'Bye-bye.';
```



```php
function validate_form() {
    global $db;
    $input = array();
    $errors = array();

    // This gets set to true only if the submitted password matches
    $password_ok = false;

    $input['username'] = $_POST['username'] ?? '';
    $submitted_password = $_POST['password'] ?? '';

    $stmt = $db->prepare('SELECT password FROM users WHERE username = ?');

    $stmt->execute($input['username']);
    $row = $stmt->fetch();
    // If there's no row, then the username didn't match any rows
    if ($row) {
        $password_ok = password_verify($submitted_password, $row[0]);
    }
    if (! $password_ok) {
        $errors[] = 'Please enter a valid username and password.';
    }

    return array($errors, $input);
}
```

Why setcookie() and session_start() Want to Be at the Top of the Page

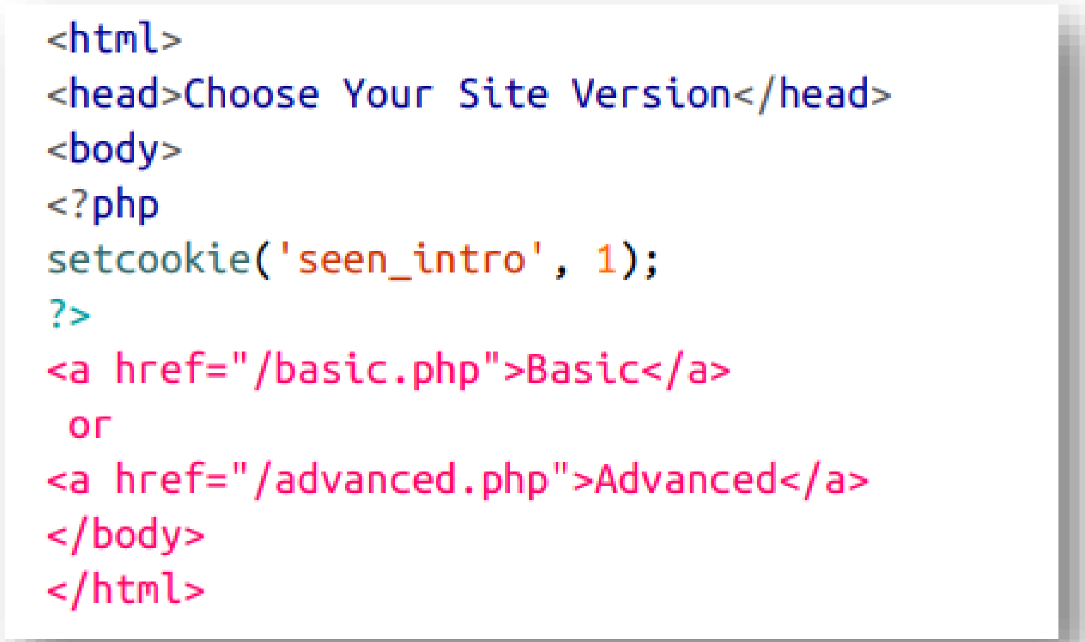
- Every HTTP communication between server and client has a header
- Header is not displayed on the webpage (screen)
- Some values in the header:
  - "this page was generated at such-and-such a time,"
  - "please don't cache this page,"
  - or "please remember that the cookie named userid has the value ralph."
- Header content must be at the beginning of the response, before the HTML body
- Once some of the body is sent—even one line—no more headers can be sent.
- setcookie() and session_start() add headers to the response, they must be added before any print.
- IF NOT, Warning!



```
Warning: Cannot modify header information - headers already sent by
(output started at /www/htdocs/catalog.php:2)
in /www/htdocs/catalog.php on line 4
```

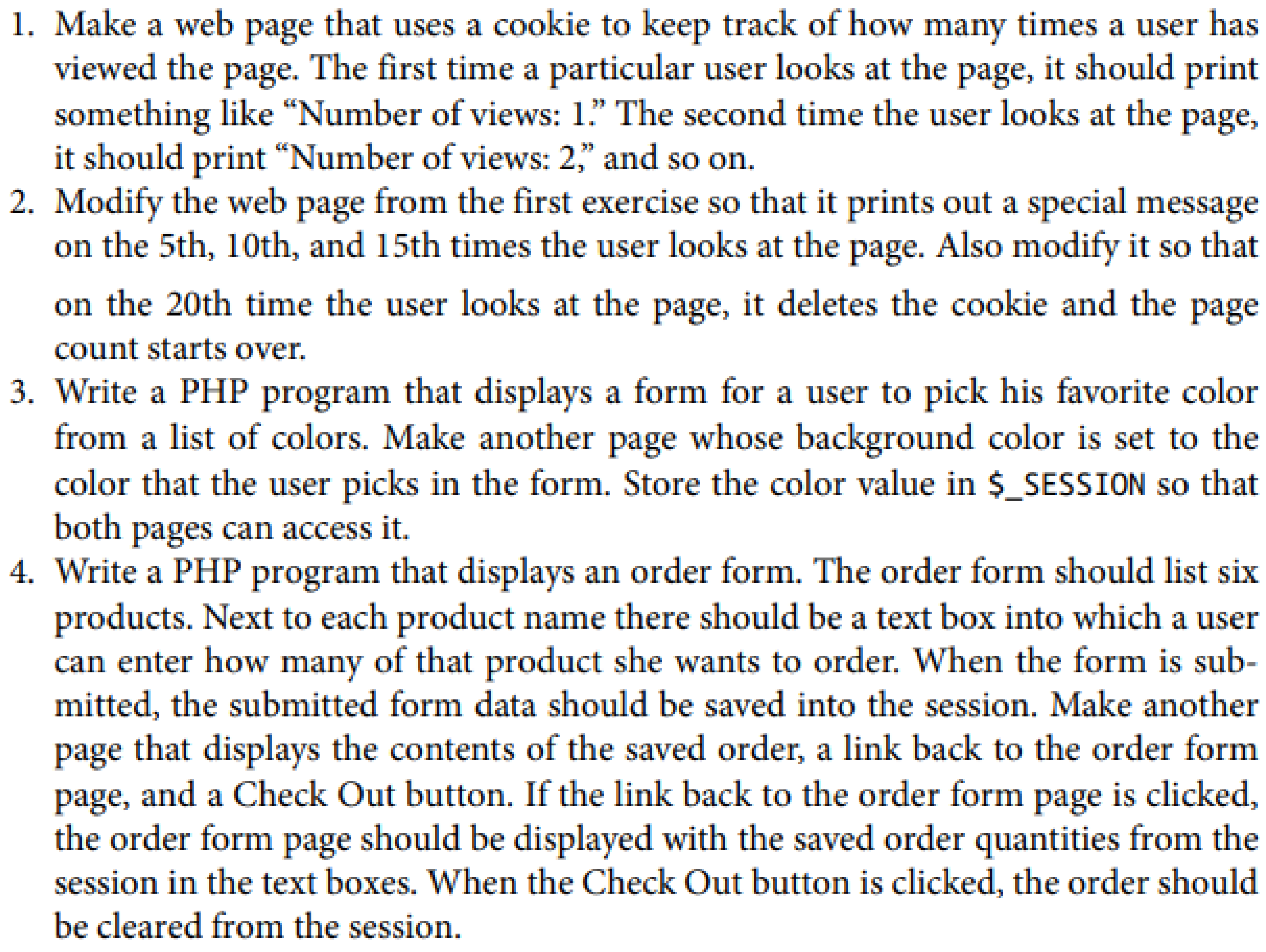
Why setcookie() and session_start() Want to Be at the Top of the Page

- An alternative: set output_buffering configuration directive to on
- This setting tells the PHP engine to wait to send any output until it's finished processing the whole request.
- Then, you can mix print statements, cookie and session functions, HTML outside of tags



```
<html>
<head>Choose Your Site Version</head>
<body>
<?php
setcookie('seen_intro', 1);
?>
<a href="/basic.php">Basic</a>
 or
<a href="/advanced.php">Advanced</a>
</body>
</html>
```

Exercises



1. Make a web page that uses a cookie to keep track of how many times a user has viewed the page. The first time a particular user looks at the page, it should print something like "Number of views: 1." The second time the user looks at the page, it should print "Number of views: 2," and so on.
2. Modify the web page from the first exercise so that it prints out a special message on the 5th, 10th, and 15th times the user looks at the page. Also modify it so that on the 20th time the user looks at the page, it deletes the cookie and the page count starts over.
3. Write a PHP program that displays a form for a user to pick his favorite color from a list of colors. Make another page whose background color is set to the color that the user picks in the form. Store the color value in `$_SESSION` so that both pages can access it.
4. Write a PHP program that displays an order form. The order form should list six products. Next to each product name there should be a text box into which a user can enter how many of that product she wants to order. When the form is submitted, the submitted form data should be saved into the session. Make another page that displays the contents of the saved order, a link back to the order form page, and a Check Out button. If the link back to the order form page is clicked, the order form page should be displayed with the saved order quantities from the session in the text boxes. When the Check Out button is clicked, the order should be cleared from the session.