PHP

Introduction to Server-Side Programming

Charles Liu

Request to a Static Site

Request to a Dynamic Site

- The server must respond dynamically if it needs to provide different client-side code depending on the situation
  - Date and time
  - Specifics of the user's request
  - Database contents – forms and authentication

You (client)

HTTP Request: GET www.facebook.com

Web server

HTTP Response: web content (HTML file)
Client-side code: HTML, CSS, JavaScript
(dynamically generated by server)

Server:

1. Look up things that go on user's profile, such as wall posts and friends → caches, database lookups
2. Generate client-side code containing these things
3. Send as HTTP response

PHP

Introduction and Basic Syntax

Charles Liu

What is PHP?

- PHP = PHP: Hypertext Preprocessor
- Server-side scripting language that may be embedded into HTML
- Ultimate goal is to get PHP files to **generate** client-side code
  - must end up with HTML, CSS, JavaScript, other client-side code!

Side-by-side

PHP File:

```
<html>
<head>
<title> PHP Introduction </title>
</head>
<body>
This is HTML! <br />
<?php
    echo 'This is PHP! <br />';
?>
</body>
</html>
```

Output: resulting HTML

```
<html>
<head>
<title> PHP Introduction </title>
</head>
<body>
This is HTML! <br />
This is PHP! <br /></body>
</html>
```

A closer look

```
<html>
<head>
        <title> PHP Introduction </title>
</head>
<body>
This is HTML! <br />
<?php
    echo 'This is PHP! <br />'; // prints to screen
    /*
    Here's a longer
    comment
    that spans multiple
    lines.
    */
?>
</body>
</html>
```

- PHP tags: `<?php and ?>`
- The `echo` command
- Single line comment (//)
- Multiple line comment (/* and */)

Viewing PHP files

- PHP files executed on the web server
- Therefore we cannot save them anywhere and view them, as with HTML files
- Must save .php files in subdirectory of web server
  - /var/www/ on many Linux configurations
  - www directory of your user directory on Athena
- Make call to web server via domain name (google.com), IP address (72.26.203.99), or localhost if on your own computer

PHP

Syntax: Variables, Operators, and Strings

Charles Liu

Variables

- Store values for future reference, use variable name to refer to the value stored in it

```
$x = 42;       // store the value 42 in $x
echo $x;       // prints 42
echo $x+1;     // prints 43, value of $x is still 42
$x = 'hello!' // type of $x can change
```

- PHP is a loosely-typed language
  - Do not need to declare the type of a variable
  - Type can change throughout the program

Operators

- Arithmetic operators
  - +, -, *, /, % (modulus – remainder after division)
- Logical AND (&&), OR (||), NOT (!)
- Assignment operators
- Shorthand for assignment operators:
  - `$x += $y` equivalent to `$x = $x + $y`
  - Also works with subtraction, multiplication, division, modulus, and string concatenation

== versus ===

- Two "equality" operators
  - == tests for "equality" in value but not necessarily type
  - === tests for "identity" in value AND type
- == ignores the distinction between:
  - Integers, floating point numbers, and strings containing the same numerical value
  - Nonzero numbers and boolean TRUE
  - Zero and boolean FALSE
  - Empty string, the string '0' and boolean FALSE
  - Any other non-empty string and boolean TRUE

Strings

- A sequence of characters
- Single and double quotes:
  - Suppose `$str = 42;`
  - `echo 'With single quotes, str is $str';`

    → **output:** `With single quotes, str is $str`
  - `echo "With double quotes, str is $str";`

    → **output:** `With double quotes, str is 42`

Strings

- Concatenation of strings – the . operator

```
$a = 'hello';
$b = 'world';
echo $a . ' ' . $b . '!';  // prints 'hello world!'
```

- String functions
  - Length: strlen()
  - Position of substring: strpos()
  - More on string functions: http://www.w3schools.com/php/php_ref_string.asp

PHP

Syntax: Conditional and Looping Statements

Charles Liu

Conditional Statements

```
if (condition / boolean expression) {
        statements
}
else if (another condition) {
        statements
}
// there may be more than one else if block
else {
        statements
}


$x = 5;
if ($x == 5) {
        echo 'The variable x has value 5!';
}
```

The while loop

```
while (condition) {
        statements
}

$x = 2;
while ($x < 1000) {
        echo $x . "n"; // \n is newline character
        $x = $x * $x;
}
```

Value of $x	$x < 1000?	Result
2	TRUE	prints 2
4	TRUE	prints 4
16	TRUE	prints 16
256	TRUE	prints 256
65536	FALSE	exits loop

The do-while loop

- The code within the loop is executed at least once, regardless of whether the condition is true

```
do {
        statements
} while (condition);
```

equivalent to:

```
statements
while (condition) {
        statements
}
```

The for loop

```
for (init; condition; increment) {
        statements
}
```

equivalent to:

```
init
while (condition) {
        statements
        increment
}
```

Prints the first 10 positive integers and their squares:

```
for ($i = 1; $i <= 10; $i++) {
        echo $i . ":" . ($i * $i) . "\n";
}
```

PHP

Syntax: Functions and Global Variables

Charles Liu

Defining your own functions

```
function function_name ($arg1, $arg2) {
        function code
        return $var   // optional
}
```

function parameters

Example: a simple multiply function

```
function multiply($x, $y) {
        echo $x * $y;
        echo "\n";
}
multiply(5, 1.2);
```
→ prints 6

```
$a = 5;
$b = 1.2;
multiply($a, $b);
```
→ prints 6

```
$a = array(1,2,3);
multiply($a, $b);
```
→ error

```
$a = "string"
multiply($a, $b);
```
→ prints 0 (?!)

Return values

- A function can return a value after it is done
  - Use this value in future computation, use like a variable, assign value to a variable
- A modified multiply function

```
function multiply($x, $y) {
        return $x * $y;
}
multiply(2,3); → prints nothing! returns value, but we don't store anywhere
echo multiply(2,3); → prints 6
$a = multiply(2,3); → assigns the value 6 to the variable $a
$b = multiply(multiply(2,3), multiply(3,4)); → assigns the value
                                                72 to the variable $b
```

Return values

- A function can return at most once, and it can only return one value
  - If it does not return anything, assignments will result in NULL
- A function ends after it returns, even if there is code following the return statement

```
function do_stuff($x) {
        if ($x % 2 == 0) {   // if even
                return $x/2   // exits function at this point
        }
        // this is ONLY executed if x is odd
        $x += 5;
        if ($x < 10) {
                $x += 3;
        }
        return x;
}
```

Making function calls

- Code inside of a function is not executed unless the function is called.
- Code outside of functions is executed whenever the program is executed.

```
<?php
… // some code
function1();      // makes function call to function1(), which
                  // in turn calls function3()

function function1() {
        … // some code
        function3();      // makes function call to function3()
}
function function2() {    // this function is never called!
        … // some code
}
function function3() {
        … // some code
}
?>
```

Variable scope

- Variables declared within a function have *local scope*
  - Can only be accessed from within the function

```
<?php
function function1() {
        … // some code
        $local_var = 5;         // this variable is LOCAL to
                                // function1()
        echo $local_var + 3;    // prints 8
}

… // some code
function1();
echo $local_var;        // does nothing, since $local_var is
                        // out of scope

?>
```

Global variable scope

- Variables declared outside a function have global scope
  - Must use global keyword to gain access within functions

```
<?php
function function1() {
        echo $a;        // does nothing, $a is out of scope
        global $a;      // gain access to $a within function
        echo $a;        // prints 4
}

… // some code
$a = 4;         // $a is a global variable
function1();

?>
```

PHP

Syntax: Arrays

Charles Liu

Arrays as a list of elements

- Use arrays to keep track of a list of elements using the same variable name, identifying each element by its *index*, starting with 0

```
$colors = array('red', 'blue', 'green', 'black', 'yellow');
```

- To add an element to the array:

```
$colors[] = 'purple';
```

- To remove an element from the array:

```
unset($colors[2]);
$colors = array_values($colors);
```

Arrays as key-value mappings

- Use arrays to keep track of a set of unique *keys* and the *values* that they map to – called an *associative array*

```
$favorite_colors = array('Joe' => 'blue', 'Elena' => 'green',
 'Mark' => 'brown', 'Adrian' => 'black', 'Charles' => 'red');
```

- To add an element to the array:

```
$favorite_colors['Bob'] = 'purple';
```

- To remove an element from the array:

```
unset($favorite_colors['Charles']);
```

- Keys must be unique:

```
$favorite_colors['Joe'] = 'purple'
```

overwrites 'blue'

Recap: arrays

- `print_r($array_name)` function lets you easily view the contents of an array
- PHP arrays as a list

```
$colors = array('red', 'blue', 'green', 'black', 'yellow');
$colors[] = purple; // add to the list

//remove 'blue' from list
unset($colors[1]);
$colors = array_values($colors);
```

- PHP arrays as a map

```
$favorite_colors = array('Joe' => 'blue', 'Elena' => 'green',
 'Mark' => 'brown', 'Adrian' => 'black', 'Charles' => 'red');
$colors['random person'] = 'white';
unset($colors['Adrian']);
```

PHP

More about arrays and the for-each loop

Charles Liu

All arrays are associative

- Take our example of a list:

```
$colors = array('red', 'blue', 'green', 'black', 'yellow');
```

- `print_r($colors)` gives:

```
Array(
        [0] => red
        [1] => blue
        [2] => green
        [3] => black
        [4] => yellow
)
```

- Turns out all arrays in PHP are associative arrays
  - In the example above, keys were simply the index into the list
- Each element in an array will have a unique key, whether you specify it or not.

Specifying the key/index

- Thus, we can add to a list of elements with any arbitrary index
  - Using an index that already exists will overwrite the value

```
$colors = array('red', 'blue', 'green', 'black', 'yellow');
$colors[5] = 'gray'; // the next element is gray
$colors[8] = 'pink';// not the next index, works anyways
$colors[7] = 'orange' // out of order works as well
```

Array functions

- `isset($array_name[$key_value])` tells whether a mapping exists AND is non-null
- `unset($array_name[$key_value])` removes the key-value mapping associated with `$key_value` in the array
  - The `unset()` function does not "re-index" and will leave gaps in the indices of a list of elements since it simply removes the key-value pairing without touching any other elements
- `array_keys($array_name)` and `array_values($array_name)` returns lists of the keys and values of the array

Adding elements without specifying the key

- Recall that we did not specify the key when adding to a list of elements:

```
$colors = array('red', 'blue', 'green', 'black',
                        'yellow');
$colors[] = 'purple';
```

- PHP automatically takes the largest integer key that has ever been in the array, and adds 1 to get the new key

```
$favorite_colors = array("Joe" => "blue", "Elena"
    => "green", "Mark" => "brown", "Adrian" =>
    "black", "Charles" => "red");
$favorite_colors[] = 'new color 1'; // key is 0
$favorite_colors[7] = 'another new color';
$favorite_colors[] = 'yet another color';  // key is 8
unset($favorite_colors[8]);
$favorite_colors[] = 'color nine'; // key is 9, the old
    // maximum is 8 even though it no longer exists!
```

The for-each loop

- The for-each loops allow for easy iteration over all elements of an array.

```
foreach ($array_name as $value) {
        code here
}
foreach ($array_name as $key => $value) {
        code here
}

foreach ($colors as $color) {
        echo $color;  // simply prints each color
}
foreach ($colors as $number => color) {
        echo "$number => $color"; // prints color with index
        // to change an element:
        // $colors[$number] = $new_color;
```

PHP

HTTP Requests and Forms

Charles Liu

Superglobals

- A few special associative arrays that can be accessed from anywhere in a PHP file
- Always $_ALLCAPS
- The $_SERVER superglobal gives information about server and client
  - $_SERVER['SERVER_ADDR'] → server IP
  - $_SERVER['REMOTE_ADDR'] → client IP
  - $_SERVER['HTTP_USER_AGENT'] → client OS and browser

Passing information to the server

- Sometimes, we require additional values be passed from client to server
  - Login: username and password
  - Form information to be stored on server
- GET request: pass information via the URL
  - http://www.yourdomain.com/yourpage.php?firstparam=firstvalue&secondparam=secondvalue
  - Access values server-side using $_GET superglobal
    - $_GET['firstparam'] => 'firstvalue'
    - $_GET['secondvalue'] => 'secondvalue'

When to use $_GET vs. $_POST

- GET requests are sent via the URL, and can thus be cached, bookmarked, shared, etc
- GET requests are limited by the length of the URL
- POST requests are not exposed in the URL and should be used for sensitive data
- There is no limit to the amount of information passed via POST

Dealing with forms

- Forms are generally used to collect data, whether the data needs to be stored on the server (registration) or checked against the server (login)
- 2 components to a form:
  - The HTML generating the form itself
  - The server-side script that the form data is sent to (via GET or POST), taking care of the processing involved
    - Server should respond appropriately, redirecting the user to the appropriate destination or generating the appropriate page

Forms: client-side

```
<html>
    <head>
        <title> A Form Example </title>
    </head><body>
<form action="welcome.php" method="post">
Name: <br /> <input type="text" name="name" /><br />
Phone Number: <br /> <input type="text" name="phone" /><br />
<input type="submit" value="Submit">
</form>
</body>
</html>
```

- form action – where to send the form data
- method – how to send the data (GET or POST)
- Name attributes become the keys used to access the corresponding fields in the $_GET or $_POST arrays

Forms: server-side

```
<html>
<head><title>This is welcome.php</title></head>
<body>
The name that was submitted was:  
<?php echo $_POST['name']; ?><br />
The phone number that was submitted was:  
<?php echo $_POST['phone']; ?><br />
</body>
</html>
```

- A simple PHP file that displays what was entered into the form
  - Can do many other things server-side depending on the situation
- Note the use of $_POST

PHP

Cookies and Sessions

Charles Liu

Cookies and sessions

- HTTP is stateless – it does not keep track of the client between requests
- But sometimes we need to keep track of this information
  - Shopping cart
  - "Remember me" on login sites
- 2 solutions to this issue
  - Cookies – small file stored client-side
  - Sessions – relevant data stored on the server

Cookies

- Cookies are stored on the user's browser, and are sent to the server on every relevant request
- The $_COOKIE superglobal makes a cookie a key-value pairing
  - Store user information as a value with a known key
  - Never assume a cookie has been set. Always check with `isset($_COOKIE[$cookie_name])` before trying to use the cookie's value

The setcookie() function

- To set a cookie in PHP:

```
setcookie(name, value, expire, path, domain);
```

- Name and value correspond to `$_COOKIE[$name] = $value`
- Expiration – cookie will no longer be read after the expiration
  - Useful to use time in seconds relative to the present:
    - `time()` + time in seconds until expiration
- Path and domain refer to where on the site the cookie is valid
  - Usually '/' for path and the top-level domain (yoursitename.com)
- To delete a cookie, set a new cookie with same arguments but expiration in the past

Setting cookies

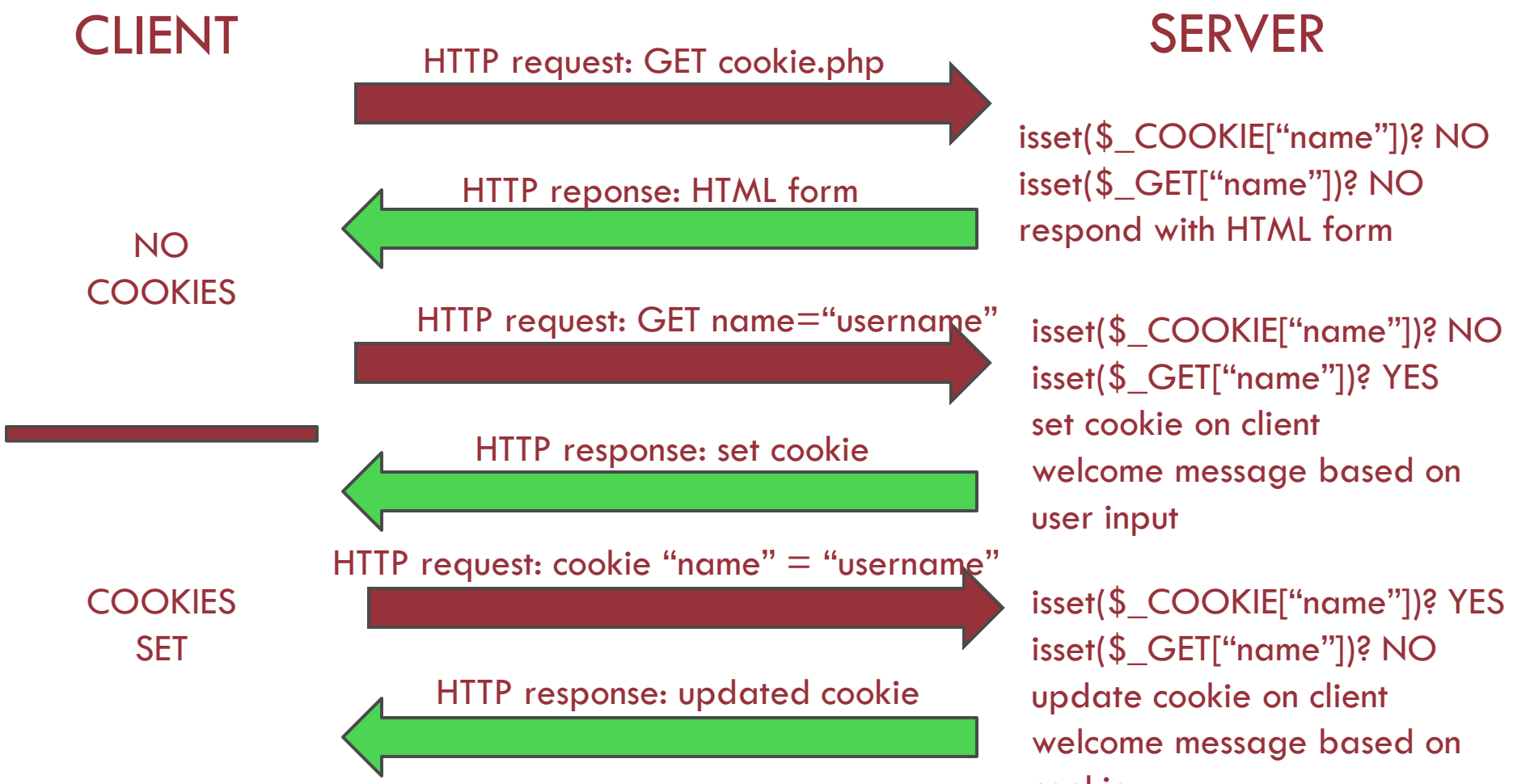
- Cookies are set via the HTTP header
  - Must be sent before the body – before any HTML, CSS, JS, etc.
- This code will not work:

```
if(isset($_COOKIE["6470"])) {
        $value = $_COOKIE['6470'];
        echo "Cookie is set to $value";
}
else {
        $value = 0;
}
// after echo statement: will not work!
setcookie("6470", $value+1, time()+60*60);?>
```

Example of cookie usage

- First visit: form with a text field for user's name
- Subsequent visits: Welcome message with the name
- Store the name field in a cookie:
  - Key: "name"; value: the user's name input into the form
- Remember: when a cookie is set (the setcookie function call is made), the cookie can only be accessed on the **next** request

Contents of the HTTP request/response



Case 1: cookies already set

```
if(isset($_COOKIE["name"])) {
        $cookie_exp = time()+60*60; // one hour
        $name = $_COOKIE["name"];
        setcookie("name", $name, $cookie_exp);
        if (isset($_COOKIE["visits"])) {
                $num_visits = $_COOKIE["visits"]+1;
                setcookie("visits", $num_visits, $cookie_exp);
        }
        echo "Welcome $name! ";
        if (isset($_COOKIE["visits"])) {
                echo "You've visited $num_visits times";
        }
}
```

Cases 2&3: first and second visits

```
// case 2: upon submission of form
else if (isset($_GET["name"])) {
    $name = $_GET["name"];
    setcookie("name", $name, $cookie_exp);
    setcookie("visits", 2, $cookie_exp);
    echo "Welcome $name! This is your second visit.";
}

// case 3: first visit: need to show form
else {
    <form action="<?php $_SERVER["PHP_SELF"] ?>" method="get">
    Enter your name here: <input type="text" name="name" />
    <br /><input type="submit" />
    </form>
}
```

Sessions

- Two main disadvantages of cookies
  - Limited in size by browser
  - Stored client-side → can be tampered with
- Sessions store user data on the server
  - Limited only by server space
  - Cannot be modified by users
- A potential downside to sessions is that they expire when the browser is closed
- Sessions are identified by a session id: often a small cookie! But the rest of the data is still stored on the server

Using sessions

- Call session_start() at top of **every** page to start session
  - Sets a cookie on the client: must follow same rules as cookies (before any HTML, CSS, JS, echo or print statements)
- Access data using the $_SESSION superglobal, just like $_COOKIE, $_GET, or $_POST

```
<?php
session_start();
if (isset($_SESSION["count"])) {
    $_SESSION["count"] += 1;
    echo "You\'ve visited here {$_SESSION['count']} times";
}
else {
    $_SESSION["count"] = 1;
    echo "You\'ve visited once";
}
?>
```

Removing sessions

- Remove an individual element of the $_SESSION superglobal
  - unset($_SESSION['key_name']);
  - The session still exists and can be modified.
- Destroy the entire session, remove all data
  - Use the function session_destroy()
  - $_SESSION no longer valid
  - Will need to call session_start() to start a new session

Recap: a comparison

	COOKIES	SESSIONS
Where is data stored?	Locally on client	Remotely on server
Expiration?	Variable – determined when cookie is set	Session is destroyed when the browser is closed
Size limit?	Depends on browser	Depends only on server (practically no size limit)
Accessing information?	$_COOKIE	$_SESSION
General use?	Remember small things about the user, such as login name. Remember things after re-opening browser	Remembering varying amount of data about the user in one browsing "session"

PHP

MySQL

Charles Liu

Databases and MySQL

- Recall the basic reason for server-side programming
  - We need to store client data or look up data stored on the server
- Databases give us an easy way to issue "commands" to insert, select, organize, and remove data
- MySQL: open-source database, relatively easy to set up, easy to use with PHP
  - Other SQL databases, as well as non-SQL options such as MongoDB

Connecting to MySQL

- MySQL database server can contain many databases, each of which can contain many tables
- Connecting to the server via PHP:

```
$db = mysql_connect(server, username, password);
if ($db) {
        // terminate and give error message
        die(mysql_error());
}
mysql_select_db(database_name, $db);
```

- $db is a database *resource type*. We use this variable to refer to the connection created

Making SQL queries

- PHP function for making queries:

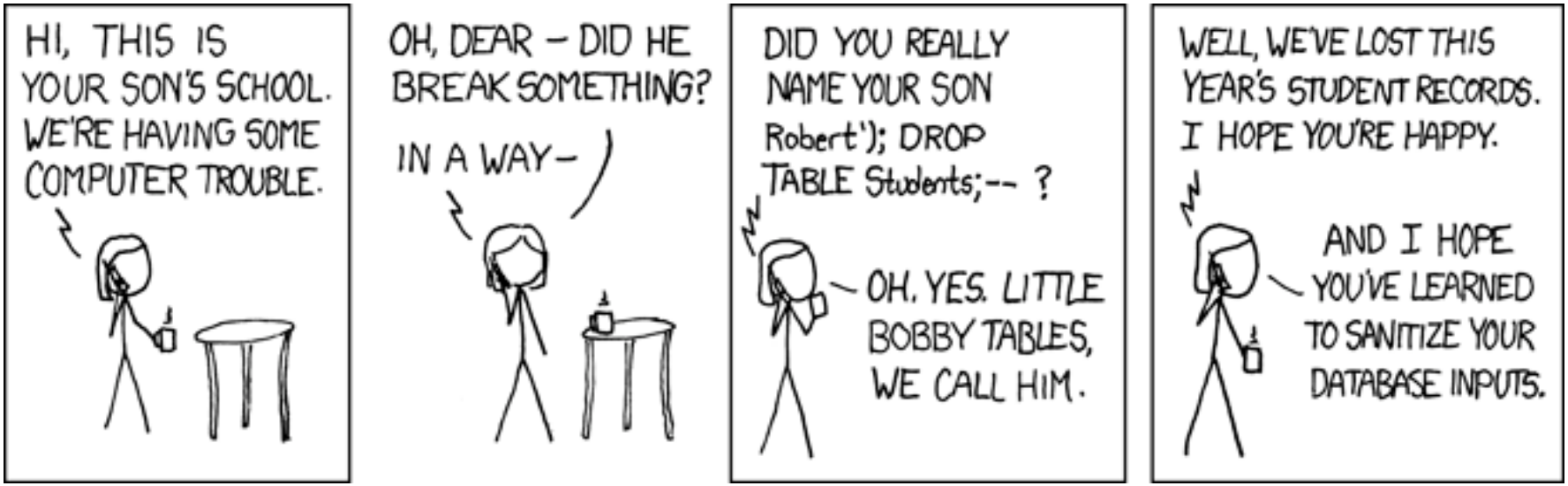
```
mysql_query(query_string, db_resource);
```

- Queries that return information, such as SELECT: returns a resource

```
$result = mysql_query(query_string, $db);
```

  - In this case, this resource is stored in the variable $result
- Other queries, returns TRUE upon success.
- All queries return FALSE on failure. Best practice is to handle the error (e.g. `die(mysql_error())`)

Never trust user input



SQL injection

- Attacker guesses the format of a query, then exploits
  - If the attacker is able to form a valid SQL query using one of the input fields, then there may be unintended results
- Look at this code which simply displays the phone number given a correct username and password

SQL injection: example

```
$db = mysql_connect("localhost", "6470user", "6470") or
        die(mysql_error());
mysql_select_db("6470example", $db) or die(mysql_error());
if (isset($_POST["username"]) && isset($_POST["password"])) {
        $user = $_POST["username"];
        $pass = $_POST["password"];
        $query = "SELECT PHONE FROM userinfo WHERE USER='$user'
                        and PASSWORD='$pass'";
        echo $query . "<br />";
        $result = mysql_query($query, $db);
        $row = mysql_fetch_assoc($result);
        if ($row) {
                echo "Phone number is: {$row['PHONE']}";
        }
        else {
                echo "Invalid user or password";
        }
}
```

SQL injection: example

- The issue here is that we are “trusting” user input.
- What if the user inserts the string

  ```
  randompass' OR '1=1
  ```

  as the password?
- Resulting query:

  ```
  SELECT PHONE FROM userinfo WHERE
  USER='username' and PASSWORD='randompass'
  OR '1=1'
  ```
- ‘1=1’ always true. We can get the server to give the phone number regardless of username/password!
- **Fix: must pass ALL user input through the function mysql_real_escape_string()**

Retrieving information from a query

- Loop over the returned $result resource, row by row
- mysql_fetch_assoc() function: turns a row of the result into key-value pairs, where keys are the names of the fields and their values are the corresponding values in the table

```
$result = mysql_query(query, $db);
while ($row = mysql_fetch_assoc($result)) {
   $col1 = $row['column_1_name'];
   $col2 = $row['column_2_name'];
   // and so forth...
}
```

A registration-login example

- Login page
  - Check username and password
  - If already logged in (use sessions!), welcome the user by name
  - Link to register page
- Register page
  - Form for registration
  - If registration is successful, confirm the username
  - Link back to login page
- Complete code can be downloaded from the video lectures website

A shared database resource

- Both login and register pages use the same database connection
- Put database connection, select database code into the same file
- Reference the connection resource ($db) in other files

```
<?php
$db = mysql_connect("localhost", "6470user", "6470") or
        die(mysql_error());
mysql_query("CREATE DATABASE IF NOT EXISTS 6470example") or
        die(mysql_error());
mysql_select_db("6470example", $db) or die(mysql_error());
mysql_query("CREATE TABLE IF NOT EXISTS users (USERNAME
        VARCHAR(2000), PASSWORD VARCHAR(2000))") or
        die(mysql_error());
?>
```

The login page – handle login request

```
if (isset($_POST["username"]) && isset($_POST["password"])) {
    require("db.php"); // establish DB connection
    $user = $_POST["username"];
    $pass = $_POST["password"];
    $query = "SELECT PASSWORD from users WHERE USERNAME='" .
        mysql_real_escape_string($user) . "'";
    $result = mysql_query($query, $db) or die(mysql_error());
    $row = mysql_fetch_assoc($result);
    if ($pass == $row["PASSWORD"]) {
        $_SESSION["username"] = $user;
    }
    else {
        echo "Invalid username or password <br />";
    }
}
```

The register page

```
if (isset($_POST["username"]) && isset($_POST["password"])) {
    require("db.php");
    $user = mysql_real_escape_string($_POST["username"]);
    $pass = mysql_real_escape_string($_POST["password"]);
    $query = "INSERT INTO users VALUES ('$user', '$pass')";
    mysql_query($query, $db) or die(mysql_error());
    echo "Registration for $user was successful <br /><br />";
    // HTML login <a href> tag
} else {
    // HTML form
}
```

MySQL recap

- Connecting to database
  - $db= mysql_connect(location, username, password)
  - mysql_select_db(db_name, $db)
- Making a query
  - $result = mysql_query(query_string, $db)
- Getting results of query
  - while($row = mysql_fetch_assoc($result))
- Sanitizing user input
  - $username = mysql_real_escape_string($_POST[“username”])

PHP

Conclusion

Charles Liu

What we've talked about…

- Purpose of server-side programming
- Basic PHP syntax, arrays, functions
- Specifics to websites: cookies, sessions, HTTP requests and forms, MySQL
- Other server-side solutions:
  - ASP.NET
  - Python
- PHP's extensive documentation: http://www.php.net/manual/en

GOOD LUCK!

nextjump Microsoft Palantir Technologies tripadvisor get the truth. then go.

Quora OOYALA facebook