PHP

Part 2

IMY 220 ● Lecture 5

Form Validation

Start with **HTML5 and JavaScript validation** in order to show error message **on the same page** as the form.

Then use **PHP validation** on the result page in case the user submitted “bad” data.

htmlspecialchars() function

The htmlspecialchars() function converts special characters to HTML entities.

This means that it will replace HTML characters like < and > with &lt; and &gt;.

This prevents attackers from exploiting the code by injecting HTML or JavaScript code (Cross-site Scripting attacks) in forms.

- Be aware that any JavaScript code can be added inside the <script> tag.

htmlspecialchars() function

Use the htmlspecialchars() function; if a user tries to submit the following in a text field:

```
<script>location.href('http://www.hacked.com')</script>
```

This would not be executed, because it would be saved as HTML escaped code, like this:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

The code is now safe to be displayed on a page or inside an e-mail.

trim & stripslashes

We will also do two more things when the user submits the form:

- trim() - strip **unnecessary characters** (extra space, tab, newline) from the user input data
- stripslashes() - **removes backslashes** (\) from the user input data (un-quotes a quoted string.)

```php
<?php
    str = "Is your name O\'reilly?";

    echo stripslashes($str);
    // Outputs: Is your name O'reilly?
?>
```

trim & stripslashes

Create a function that will do all the checking for you (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each $_POST variable with the test_input() function.

trim & stripslashes

```php
<?php
$name = test_input($_POST["name"]);
$email = test_input($_POST["email"]);
$website = test_input($_POST["website"]);
$comment = test_input($_POST["comment"]);
$gender = test_input($_POST["gender"]);

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>
```

PHP date()

The PHP date() function can be used to display the current date in a specific format.

The PHP date() function formats a *timestamp* to a more readable date and time.

Syntax:

```
date(format, timestamp);
```

PHP date()

Format parameter (required)

The first parameter in the date() function specifies how to format the date/time.

It uses **letters** to represent date and time formats.

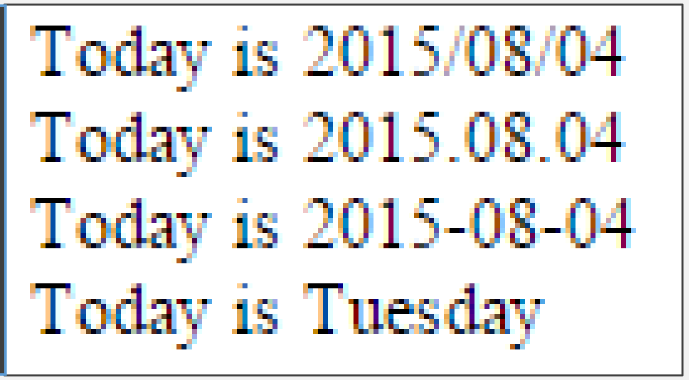
Other characters, like"/", ".", or "-" can also be inserted between the letters to add additional formatting.

PHP date()

Letters to format the day:

- d - The day of the month (01-31)
- m - The current month, as a number (01-12)
- Y - The current year in four digits
- l (lowercase 'L') - Represents the day of the week

```php
<?php
    echo "Today is " . date("Y/m/d") . "<br/>";
    echo "Today is " . date("Y.m.d") . "<br/>";
    echo "Today is " . date("Y-m-d") . "<br/>";
    echo "Today is " . date("l") . "<br/>";
?>
```



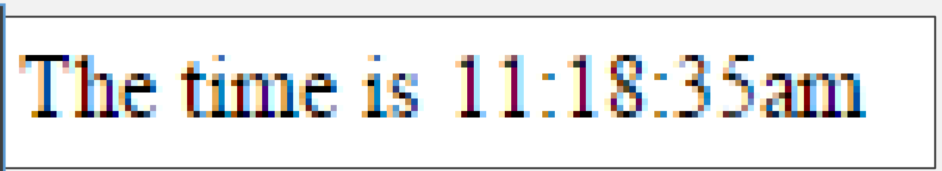
```
Today is 2015/08/04
Today is 2015.08.04
Today is 2015-08-04
Today is Tuesday
```

PHP date()

Letters to format the time:

- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

```
<?php
   echo "The time is " . date("h:i:sa");
?>
```



```
The time is 11:18:35am
```

Read up on Datetime formats here: http://php.net/manual/en/function.date.php

Server Side Includes (SSI)

You can insert the content of a file into a PHP file before the server executes it, with the include or require statements.

The two statements are identical in every way, except how they handle errors.

Server Side Includes (SSI)

The include statements generates a warning (but the script will continue execution)

The require statements generates a fatal error (and the script execution will stop after the error).

These two functions are used to create functions, headers, footers, or elements that can be reused on multiple pages.

Server Side Includes (SSI)

Saves time – DRY approach

This means that you can create a standard header or menu file that you want all your web pages to include.

When the header needs to be updated, you can only update this one include file, or when you add a new page to your site, you can simply change the menu file (instead of updating the links on all web pages).

Server Side Includes (SSI)

Some advantages of DRY code:

- Write less code
- Maintainability: only have to make changes in one place
- Readability: by centralising and naming functionality, we create more readable code

The include statement

Assume that you have a standard header file, called “*footer.php*".

To include the header file in a page, use the include statements, like this:

```
<html>
<body>
   <h1> Welcome to my home page! </h1>
   <p> Some text. </p>
   <?php
      include "footer.php";
   ?>
</body>
</html>
```

The include statement

The include statements takes all the text in a specified file and copies it into the file that uses the include function.

```php
// vars.php
<?php
   $colour = "green";
   $fruit = "apple";
?>
```

```php
// test.php
<?php
    echo "A $colour $fruit";
    // Output: Notice: Undefined
    // variable: colour
    // Output: A

    include "vars.php";

    echo "A $colour $fruit";
    // Output: A green apple

?>
```

The require statement

The require statement is identical to include, except that it handles errors differently.

The include statement generates a warning (but the script will continue execution) while the require statement generates a fatal error (and the script execution will stop after the error).

The require statement

It is recommended to use the require statement instead of include, because scripts should not continue executing if files are missing or misnamed.

Variable scope

Quick word about variable scope: variable cannot be used in functions they have not been defined in, for example

```
$name = "Diffie";

function sayName(){
     echo $name;
}

// This does not print out anything
sayName();
```

Variable scope

To be able to use the variable inside the function, we must make it global using the global keyword.

```php
$name = "Diffie";

function sayName(){
     global $name;
     echo $name;
}

// Now this prints out "Diffie"
sayName();
```

PHP Cookies

A cookie is often used to identify a user.

A cookie is a small file that the server embeds on the user's computer.

Each time the same computer requests a page with a browser, it will send the cookie too.

With PHP, you can both create and retrieve cookie values.

Baking a Cookie

Syntax:

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

Baking a Cookie

Example: Create a cookie named "user" and assign the value “*Gumball Watterson*" to it. We also specify that the cookie should expire after one hour:

Note: The setcookie() function must appear BEFORE the <html> tag.

```
<?php
   setcookie("user", "Gumball Watterson", time()+3600);
?>

<html>
   <body>
   </body>
</html>
```

Retrieving a Cookie Value

The PHP $_COOKIE variable is used to retrieve a cookie value.

Example: Retrieve the value of the cookie named "user" and display it on a page:

```
<?php
    // Print a cookie
    echo $_COOKIE["user"];

    // A way to view all cookies
    print_r($_COOKIE);
?>
```

Test if a cookie is set

Use the isset() function to find out if a cookie has been set:

```
<html>
<body>
   <?php
      if(isset($_COOKIE["user"]))
         echo "Welcome " . $_COOKIE["user"] . "!<br/>";
      else
         echo "Welcome guest!<br/>";
   ?>
</body>
</html>
```

Deleting a Cookie

When deleting a cookie you should set the expiration date to the past.

```
<?php
    // set the expiration date to one hour ago
    setcookie("user", "", time() - 3600);
?>
```

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the user’s computer.

PHP Sessions

When you are working with an application, you open it, do some changes and then you close it.

- This is much like a Session.

The computer knows who you are.

- It knows when you start the application and when you end.

But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

PHP Sessions

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc.).

However, session information is temporary and will be deleted after the user has left the website.

If you need a permanent storage you may want to store the data in a database.

PHP Sessions

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID.

The UID is either stored in a cookie or is propagated in the URL.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session.

```
<?php
   session_start();
?>

<html>
   …
```

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

Storing a Session Variable

The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
   session_start();
   // store session data
   $_SESSION["views"]=1;
?>

<html>
<body>
<?php
   // retrieve session data
   echo "Pageviews = " . $_SESSION["views"];
?>
</body>
</html>
// Output: Pageviews = 1;
```

Session

To show all the session variable values for a user session:

```
<?php
   print_r($_SESSION);
?>
```

Session Example

Create a simple page-views counter.

The isset() function checks if the "views" variable has already been set.

If "views" has been set, we can increment our counter.

If "views" doesn't exist, we create a "views" variable, and set it to 1.

Session Example

```php
<?php
   session_start();
   if(isset($_SESSION["views"]))
      $_SESSION["views"] = $_SESSION["views"] + 1;
   else
      $_SESSION["views"] = 1;
   echo "Views = " . $_SESSION["views"];
?>
```

Session Example

Note: This is very useful to prevent users from accessing web pages when they are either not registered or logged in. Check for their session, if it does not exist, redirect them to the log in page.

Destroying a Session

If you wish to delete some session data, you can use the session_unset() and the session_destroy() function.

```php
<?php
    // remove all session variables
    session_unset();

    // destroy the session
    session_destroy();
?>
```

PHP File Upload

To allow users to upload files from a form can be very useful.

Note: make sure your PHP is configured to allow file uploads by setting file_uploads to On in php.ini

```
<!DOCTYPE html>
<html>
<body>
   <form action="upload.php" method="post" enctype="multipart/form-data">
      Select image to upload:
      <input type="file" name="fileToUpload" id="fileToUpload" />
      <input type="submit" value="Upload Image" name="submit" />
   </form>
</body>
</html>
```

PHP File Upload

Notice the following about the HTML form above:

- The enctype attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded.

- The type="file" attribute of the <input> tag specifies that the input should be processed as a file. For example, when viewed in a browser, there will be a browse-button next to the input field.
  - Note: Allowing users to upload files is a big security risk. Only permit trusted users to perform file uploads.

Create the Upload Script

The “upload.php” file:

```
$target_dir = "uploads/";
$uploadFile = $_FILES["fileToUpload"];
$target_file = $target_dir . basename($uploadFile["name"]);
$imageFileType = pathinfo($target_file, PATHINFO_EXTENSION);
// Check if image is an actual image (Note that this method is
unsafe)
if(isset($_POST["submit"])){
      $check = getimagesize($uploadFile["tmp_name"]);
      if($check !== false){
            echo "File is an image - " . $check["mime"] . ".";
      }
      else {
            echo "File is not an image.";
      }
}
```

Create the Upload Script

$target_dir = "uploads/" - specifies the directory where the file is going to be placed

$uploadFile refers to the file being uploaded

$target_file specifies the path of the file to be uploaded

$imageFileType holds the file extension of the file

Next, check if the image file is an actual image or a fake image

Create the Upload Script

By using the global PHP $_FILES array you can upload files from a client computer to the remote server.

This is a very simple way of uploading files.

For security reasons, you should add restrictions on what the user is allowed to upload.

Create the Upload Script

The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp_name" or "error". Like this:

- $_FILES["fileToUpload"]["name"] - the name of the uploaded file
- $_FILES["fileToUpload"]["type"] - the type of the uploaded file
- $_FILES["fileToUpload"]["size"] - the size in bytes of the uploaded file
- $_FILES["fileToUpload"]["tmp_name"] - the name of the temporary copy of the file stored on the server
- $_FILES["fileToUpload"]["error"] - the error code resulting from the file upload

Restrictions on Upload

In this script we add some restrictions to the file upload.

The user may only upload .gif or .jpeg files and the file size must be under 20 kb.

Restrictions on Upload

```php
<?php
   $uploadFile = $_FILES["fileToUpload"];
   if(($uploadFile["type"] == "image/gif"
   || $uploadFile["type"] == "image/jpeg"
   || $uploadFile["type"] == "image/pjpeg")
   && $uploadFile["size"] < 20000){
      if($uploadFile["error"] > 0){
         echo "Error: " . $uploadFile["error"] . "<br/>";
      }  else {
         echo "Upload: " . $uploadFile["name"] . "<br/>";
         echo "Type: " . $uploadFile["type"] . "<br/>";
         echo "Size: " . ($uploadFile["size"] / 1024) . "Kb<br/>";
         echo "Stored in: " . $uploadFile["tmp_name"];
      }
   }  else {
      echo "Invalid file";
   }
?>
```

Saving the Uploaded File

The examples above create a temporary copy of the uploaded files in the PHP temp folder on the server.

The temporary copied files disappear when the script ends.

To store the uploaded file we need to copy it to a different location.

Saving the Uploaded File

The script checks if the file already exists, if it does not, it copies the file to the specified folder.

Note: This example saves the file to a new folder called "upload"

```
if(move_uploaded_file($uploadFile["tmp_name"], $target_file)){
   echo "The file " . basename($uploadFile["name"]) . "has been uploaded.";
}  else {
   echo "Sorry, there was an error uploading your file.";
}
```

```php
<?php
    $uploadFile = $_FILES["fileToUpload"];
    if(($uploadFile["type"] == "image/gif"
    || $uploadFile["type"] == "image/jpeg"
    || $uploadFile["type"] == "image/pjpeg")
    && $uploadFile["size"] < 20000){
       if($uploadFile["error"] > 0){
          echo "Error: " . $uploadFile["error"] . "<br/>";
       }  else {
          echo "Upload: " . $uploadFile["name"] . "<br/>";
          echo "Type: " . $uploadFile["type"] . "<br/>";
          echo "Size: " . ($uploadFile["size"] / 1024) . "Kb<br/>";
          echo "Temp file: " . $uploadFile["tmp_name"];

          if(file_exists("upload/" . $uploadFile["name"])){
             echo $uploadFile["name"] . " already exists.";
          }  else  {
             move_uploaded_file($uploadFile["tmp_name"],
             "upload/" . $uploadFile["name"]);
             echo "Stored in: " . "upload/" . $uploadFile["name"];
          }
       }
    }  else {
       echo "Invalid file";
    }
?>
```

Saving the Uploaded File

Note that the code in the previous slide is not very DRY when checking for allowed file types

A better solution to check for a list of allowed file types would be something like this: https://stackoverflow.com/questions/10456113/check-file-extension-in-upload-form-in-php

Uploading multiple files

Uploading multiple files requires a few changes in your HTML and PHP

First, the (file) input name has to be defined as an array and must have an HTML5 “multiple” attribute

```
<input type="file" name="picToUpload[]" multiple="multiple" />
```

Uploading multiple files

In your PHP, you need to loop through and access the individual files being uploaded:

```
$uploadFile = $_FILES["picToUpload"];
// Profile pic is being updated
$numFiles = count($uploadFile["name"]);

for($i = 0; $i < $numFiles; $i++){
   // Use $uploadFile[parameter][index] to access individual files
   // For example: $name = $uploadFile["name"][$i];
}
```

PHP File Handling

The fopen() function is used to open files in PHP.

The first parameter of this function contains the name of the file to be opened

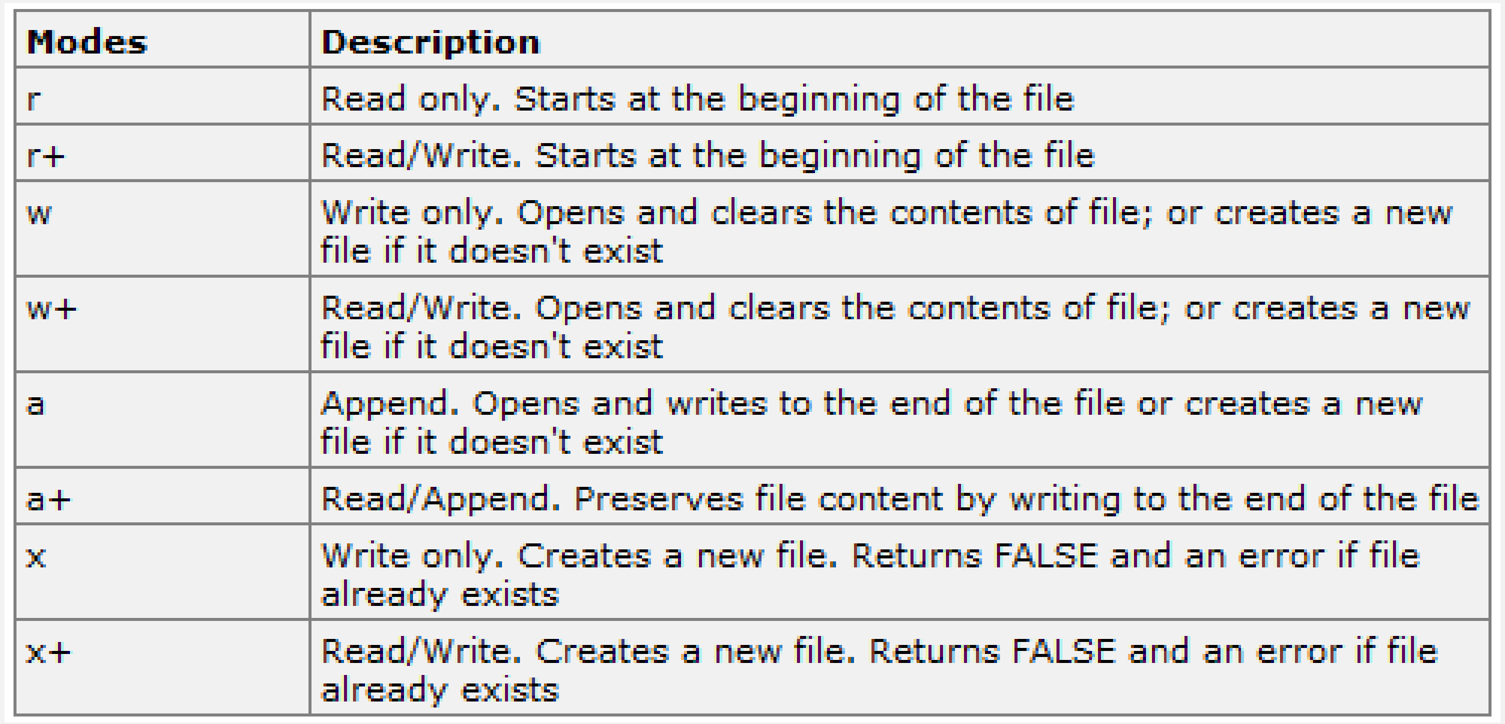
The second parameter specifies in which mode the file should be opened:

```
<html>
<body>
   <?php
      $file = fopen("welcome.txt", "r");
   ?>
</body>
</html>
```

PHP File Handling

The file may be opened in one of the following modes:

Note: If the fopen() function is unable to open the specified file, it returns 0 (false).



Modes	Description
r	Read only. Starts at the beginning of the file
r+	Read/Write. Starts at the beginning of the file
w	Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist
w+	Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist
a	Append. Opens and writes to the end of the file or creates a new file if it doesn't exist
a+	Read/Append. Preserves file content by writing to the end of the file
x	Write only. Creates a new file. Returns FALSE and an error if file already exists
x+	Read/Write. Creates a new file. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile, filesize("webdictionary.txt"));
```

Closing a File

The fclose() function is used to close an open file:

```php
<?php
   $file = fopen("test.txt", "r");

   // some code to be executed

   fclose($file);
?>
```

Check End of File

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

Note: You cannot read from files opened in w, a, and x mode!

```
if(feof($file)) echo "End of file";
```

Reading a File Character by Character

The fgetc() function is used to read a single character from a file.

Note: After a call to this function the file pointer moves to the next character.

Read a file character by character, until the end of file is reached:

```
while(!feof($file)){
   echo fgetc($file);
}
fclose($file);
```

PHP File Handling Example

The following example generates a message if the fopen() function is unable to open the specified file:

```
<?php
   $myfile = fopen("webdictionary.txt", "r") || die("Unable to open file");
   echo fread($myfile, filesize("webdictionary.txt"));
   fclose($myfile);
?>
```

Reading a File Line by Line

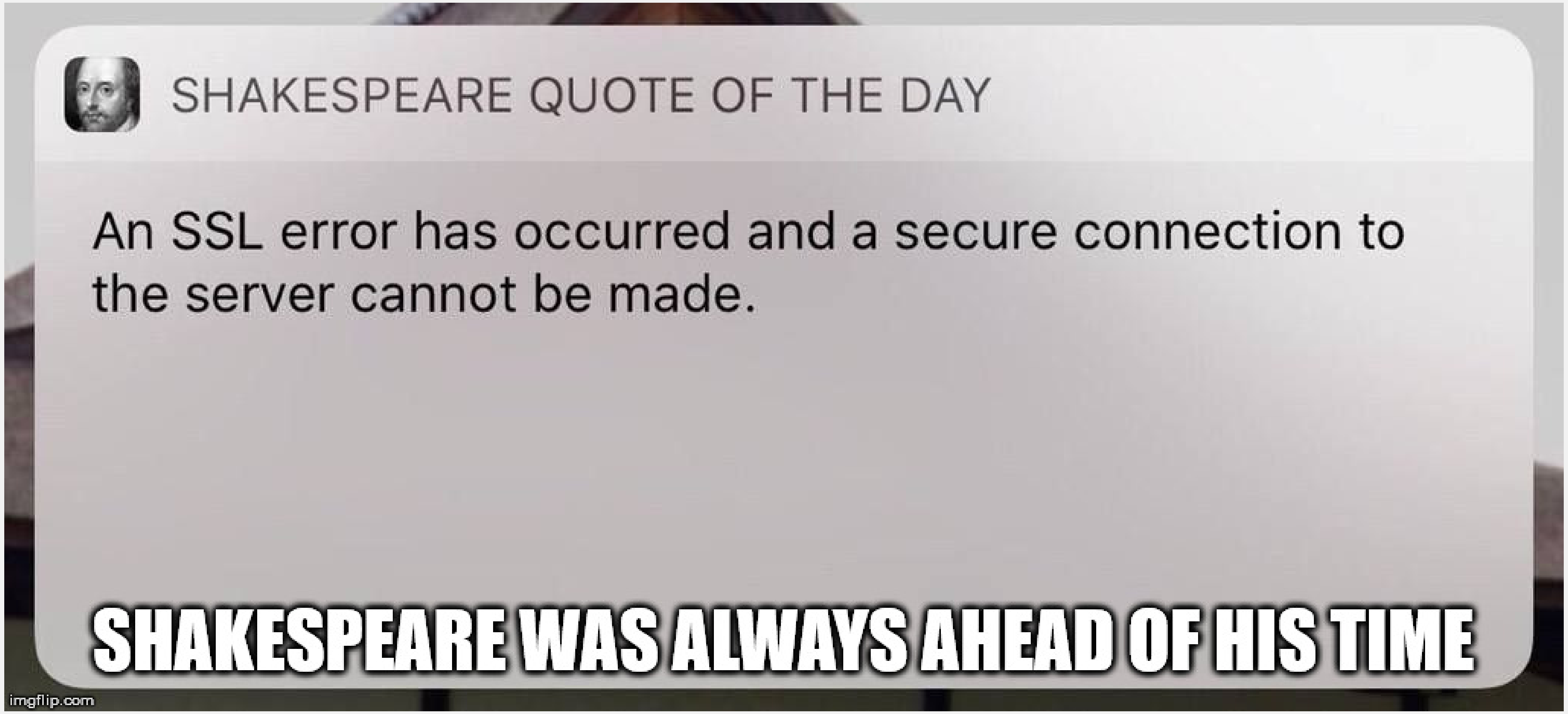
The fgets() function is used to read a single line from a file.

Note: After a call to this function the file pointer has moved to the next line.

Read a file line by line until the end of the file:

```
while(!feof($file)){
   echo fgets($file) . "<br/>";
}
fclose($file);
```

The End


SHAKESPEARE QUOTE OF THE DAY

An SSL error has occurred and a secure connection to the server cannot be made.