

# Form checking, classes, and files

INFO/CS 2300:  
Intermediate Web Design and  
Programming

# Checking user input

# This is a bad idea

```
<?php
    $user_input = $_POST[ 'user_input' ] );
    print( $user_input );
?>
```

# Be skeptical

You should *always, always, always* check  
user input

Why?

Malicious users might be  
trying to do something

Clueless users might do stuff  
you don't expect like enter  
letters instead of numbers


## form.php

```
<?php
    //Try entering these for usernames in different browsers
    //<script>window.open("http://cornell.edu");</script>
    //<p style="font-size: 5em;">steve</p>
    if ( ! empty( $_POST[ "username" ] ) ) {
        $user = $_POST[ 'username' ];
        print("<p>Welcome, $user! </p>");
    } else {
?>
        <form method="post">
            <p>What is your name?
                <input type="text" name="username">
            </p>
            <input type="submit" value="Click to submit">
        </form>
    <?php
    }
?>
```

# HTML Entities

```
print( "&lt;p&gt;steve&lt;/p&gt;" );  
      <p>steve</p>
```

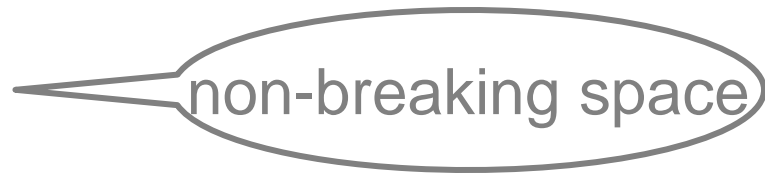
```
print( "&lt;p&gt;steve&lt;/p&gt;" );  
      <p>steve</p>
```



The diagram illustrates the mapping of HTML entities to their corresponding tags. Four arrows point from the entities in the string "&lt;p&gt;steve&lt;/p&gt;" to the corresponding characters in the rendered HTML "<p>steve</p>". Specifically, a blue arrow points from "&lt;" to "<", a magenta arrow points from "&gt;" to ">", another blue arrow points from "&lt;" to "<", and a final magenta arrow points from "&gt;" to ">".

# A few HTML Entities

&	&amp;
<	&lt;
>	&gt;
	&nbsp;
»	&raquo;
¼	&frac14;
©	&copy;



Lots more

[http://www.w3schools.com/charsets/ref\\_html\\_entities\\_4.asp](http://www.w3schools.com/charsets/ref_html_entities_4.asp)

## safer-form.php

```
<?php
    //Try entering these for usernames in different browsers
    //<script>window.open("http://cornell.edu");</script>
    //<p style="font-size: 5em;">steve</p>
    if ( ! empty( $_POST[ "username" ] ) ) {
        $username = htmlentities( $_POST[ 'username' ] );
        print( "<p>Welcome, $username!</p>" );
    } else {
?>
        <form method="post">
            <p>What is your name?
                <input type="text" name="username">
            </p>
            <input type="submit" value="Click to submit">
        </form>
    <?php
    }
?>
```



# filter\_input

```
<?php
$user = htmlentities( $_POST[ 'username' ] );
print("<p>Welcome, $user! </p>");
```

also INPUT\_GET

post variable  
name

//Alternatively

```
$user = filter_input( INPUT_POST, 'username',
    FILTER_SANITIZE_FULL_SPECIAL_CHARS)
```

```
?>
```

This filter constant is  
not as thorough as  
htmlentities

# Input Numbers

```
<?php
    $number_input = filter_input( INPUT_POST, 'number',
                                   FILTER_VALIDATE_INT );

    //Check to see if a number or something else
    if( is_numeric( $number ) ) {
        print( "<p>Your number is $number!</p>" );
    } else {
        print( "<p>You didn't enter a number</p>" );
    }

?>
```

# More filter constants

- `FILTER_VALIDATE_INT`
- `FILTER_VALIDATE_FLOAT`
- `FILTER_VALIDATE_URL`
- `FILTER_VALIDATE_EMAIL`

<http://php.net/manual/en/filter.constants.php>

# Input Dates

Prevent clueless entries by using HTML5 date input, but not for IE and Firefox

```
<input type="date" name="birthday">
```

Validating dates is complicated.

This is a good place to start

<http://stackoverflow.com/questions/10691949/>

# Being careful

- Use `preg_match` and / or `filter_input` to check that input contains only what you want.
- If you want to print out a string the user entered, use `htmlentities($input)` to make sure all HTML special characters are translated.

# Click in!

The purpose of htmlentities is:

- A. To check user input for unwanted characters
- B. To safely print text to the screen
- C. To remove unwanted characters
- D. All of the above
- E. None of the above

# Click in!

The purpose of htmlentities is:

- A. To check user input for unwanted characters
- B. To safely print text to the screen**
- C. To remove unwanted characters
- D. All of the above
- E. None of the above

# PHP Classes



# If we talk about a movie ...

We might describe its

- Name
- Year released
- Length
- and many other qualities

# Example – Movie class

```
<?php
    class Movie {
        public $title;
        public $year;
        public $length;
    }

    $movie = new Movie();
    $movie->title = 'The Princess Bride';
    $movie->year = '1987';
    $movie->length = 98;

    print( "Have you watched $movie->title?" );
?>
```

# Movie class with a function

```
<?php
```

```
class Movie {  
    public $title;  
    public $year;  
    public $length;  
  
    public function the_question() {  
        return "Have you watched $this->title?";  
    }  
}
```

```
$movie = new Movie();  
$movie->title = 'The Princess Bride';  
$movie->year = '1987';  
$movie->length = 98;
```

```
print( $movie->the_question() );
```

```
?>
```

# What is '\$this'?

`$this` is a special variable used inside method definitions to refer to the current instance of the object.

# Using a class more than once

```
<?php
    class Movie {
        public $title;
        public $year;
        public $length;
    }
    $movie_1 = new Movie();
    $movie_1->title = 'The Princess Bride';
    $movie_1->year = '1987';
    $movie_1->length = 98;

    $movie_2 = new Movie();
    $movie_2->title = 'Finding Nemo';
    $movie_2->year = '2003';
    $movie_2->length = 100;

    print( "Have you seen $movie_1->title or $movie_2->title?" );
?>
```

# Objects and Instances

Objects are a group of **variables and functions** that can work easily together.

When we use the object ( \$movie\_1 and \$movie\_2) we call it an **instance of the object**.

# Properties and Methods

```
class ObjectName {  
    public variablename1;  
    public variablename2;  
    ...
```



variables = properties

```
    function function_name() {  
    }  
  
    ...  
}
```



functions = methods

# A special method: constructors

```
class Movie {  
    public $title;  
    public $year;  
    public $length;  
  
    function __construct( $title = "", $year = "", $length = null ) {  
        $this->title = $title;  
        $this->year = $year;  
        $this->length = $length;  
    }  
}
```

```
$movie = new Movie( 'The Princess Bride', '1987', 98 );  
print( "Have you watched $movie->title?" );
```



# A new kind of variable

Defining an object defines a new *kind* of variable (e.g. like a string, or integer).

Just like `$a = 5` and `$b = 10` are both of the integer variable type, `$movie_1` and `$movie_2` are both of the Movie variable type.

# OOP

Some programming languages/styles  
work entirely with objects: said to be  
*object-oriented* (e.g. Java, C++, Ruby)

OOP = object-oriented programming

# Why use objects?

Code organization – it is pretty clear from the file structure where variables and functions for movies belong

Namespace – a generic function name like `fix_title` won't collide with a function of the same name somewhere else in the code

Easier to pass around groups of variables

# Files

Sometimes you'll want to a website to remember information between visits. One way to do this is to store information in a file.



# File functions

`file_exists( 'filename' )`

Checks whether a file or directory exists

Returns TRUE or FALSE

# File functions open/close

`fopen(filename, mode)`

Opens the file *filename* for reading and/or writing depending on **mode**

Returns a **file pointer** if file is opened successfully, or “false” if not

`fclose( file pointer )`

Clean up when done

# fopen modes

```
$file_pointer = fopen("rollercoaster.txt", $mode);
```

Mode	Read	Write	Overwrite	Create	Pointer
r	X				beginning
r+	X	X			beginning
w		X	X	X	beginning
w+	X	X	X	X	beginning
a		X		X	end
a+	X	X		X	end

more >> <http://php.net/manual/en/function.fopen.php>



# Click In!

What mode should be used to log error messages in order as they occur?

A. r+

B. w

C. w+

D. a

E. a+

Mode	Read	Write	Overwrite	Create	Point
r	X				beg
r+	X	X			beg
w		X	X	X	beg
w+	X	X	X	X	beg
a		X		X	end
a+	X	X		X	end

# Click In!

What mode should be used to log error messages in the order they occur?

A. r+

B. w

C. w+

D. a

E. a+

Mode	Read	Write	Overwrite	Create	Point
r	X				beg
r+	X	X			beg
w		X	X	X	beg
w+	X	X	X	X	beg
a		X		X	end
a+	X	X		X	end

# Writing files

`fputs($file_pointer, $string)`

Writes \$string to the file given by  
filepointer \$file\_pointer;

returns “false” if there’s an error

# Reading files

`fgets( $file_pointer )`

Returns the next line from the file given by  
filepointer \$file\_pointer

`feof( $file_pointer )`

Returns true if the end of file has been  
reached

# A quick way to read a file

```
$file_pointer = fopen( 'file.txt', 'r' );  
if ( ! $file_pointer ) { print( 'error' ); exit; }  
$lines = array();  
while( ! feof( $file_pointer ) ) {  
    $lines[] = fgets( $file_pointer );  
}  
fclose( $file_pointer );
```

# An even quicker way

```
$myarray = file( "rollercoaster.txt" );
```

# Preparing a row for storage

In our PHP program:

```
$row = array( "Top Thrill", "Steel", "Cedar Point", 8 );
```

In file:

```
Top Thrill \t Steel \t Cedar Point \t 8 \n
```



Now you try...



# Saving and reading the data

Suppose the data in the table is stored as an array `$rides`, where each element is another array, whose first element is the first field for that row (e.g. “El Toro”), second element is the second field (e.g. “Steel”), etc.

Write the code that opens the file “rollercoaster.txt” and writes out the table.

What about reading the file back in?

# One idea

```
$rides = array(
    array( 'Top Thrill Dragster', 'Steel', 'Cedar Point', 8 );
    array( 'El Toro', 'Steel', 'Six Flags Great Adventure', 1);
);
$lines = array();
foreach( $rides as $ride ) {
    $line = implode( "\t", $ride );
    $lines[] = $line;
}
$contents = implode( "\n", $lines );
$file_pointer = fopen( 'rollercoaster.txt', 'w' );
fputs( $file_pointer, $contents );
fclose( $file_pointer );
```

# Restoring the rides array?

```
$file_pointer = fopen( 'rollercoaster.txt', 'r' );  
$rides = array();  
while( ! feof( $file_pointer ) ) {  
    $line = fgets( $file_pointer );  
    $ride = explode( "\t", $line );  
    //Strip the "\n" off the end of the value  
    $ride[3] = intval( $ride[3] );  
    $rides[] = $ride;  
}
```

# Review

- You must *always, always, always* check your user's input
- Objects are useful for 'packaging' up functions and associated data.
- You can read and write files for storing data between page loads

# Reminders...

- Project 1 due Tuesday 5 pm