INFO I-CE9224: Introduction to PHP Programming

Session 6
July 18, 2012

Resources

http://davehauenstein.com/nyu/INFO1-CE9224-2012-Summer

Username: nyuscps
Password: $nyuscps$

Class 6 Agenda

- Review Functions
  - Pass arguments by reference
  - Anonymous Functions (aka Closures)
- Server Side Validation
  - Versus Client Side Validation
  - Using isset() and empty()
- Escaping input data
- Complex Validation
  - Regular Expressions
  - Date Validation

Review Pass by Reference

Passing Arguments by Reference

- We've learned how to pass arguments by value.
- Function can now alter original value.
- After function call, variable is modified without a return statement or assignment expression.
- Place an ampersand (&) before argument in function declaration.

Passing By Value

```php
function arrayValuesToUpper($values)
{
    foreach($values as $key => $value) {
        $values[$key] = strtoupper($value);
    }

    return $values;
}

$cars = array('Ford', 'Chevy', 'Dodge');

// Scenario 1
arrayValuesToUpper($cars);
print_r($cars);

// Scenario 2
$cars = arrayValuesToUpper($cars);
print_r($cars);
```

Passing By Value

```php
function arrayValuesToUpper($values)
{
    foreach($values as $key => $value) {
        $values[$key] = strtoupper($value);
    }

    return $values;
}

$cars = array('Ford', 'Chevy', 'Dodge');

// Scenario 1
arrayValuesToUpper($cars);
print_r($cars);

// Scenario 2
$cars = arrayValuesToUpper($cars);
print_r($cars);
```

Important

Passing By Value

```
Array
(
    [0] => Ford
    [1] => Chevy
    [2] => Dodge
)
Array
(
    [0] => FORD
    [1] => CHEVY
    [2] => DODGE
)
```

Passing By Value

- Scenario 1 - calling arrayValuesToUpper does not modify $cars. $cars holds the same value as it was originally assigned.
- Scenario 2 - assigning the result of the function call, arrayValuesToUpper, to $cars finally modifies the value of $cars.

Passing By Reference

```php
function arrayValuesToUpper(&$values)
{
    foreach($values as $key => $value) {
        $values[$key] = strtoupper($value);
    }
}

$cars = array('Ford', 'Chevy', 'Dodge');

// Scenario 1
arrayValuesToUpper($cars);
print_r($cars);

// Scenario 2
$cars = arrayValuesToUpper($cars);
print_r($cars);
```

Passing By Reference

```php
function arrayValuesToUpper(&$values)
{
    foreach($values as $key => $value) {
        $values[$key] = strtoupper($value);
    }
}
```

```php
$cars = array('Ford', 'Chevy', 'Dodge');

// Scenario 1
arrayValuesToUpper($cars);
print_r($cars);

// Scenario 2
$cars = arrayValuesToUpper($cars);
print_r($cars);
```

Important

Passing By Reference

```
Array
(
    [0] => FORD
    [1] => CHEVY
    [2] => DODGE
)


NULL
```

Passing By Reference

- Scenario 1 - calling arrayValuesToUpper DOES modify $cars. $cars holds the UPDATED value. It was modified by the function.

- Scenario 2 - assigning the result of the function call, arrayValuesToUpper, to $cars sets $cars to NULL. Why?

- Because... If a function does not have a return statement, by default it returns NULL. That is why the second $cars prints out NULL.

Passing By Reference

PHP built-in array functions,

sort() and rsort(), use pass by reference.

```php
$cars = array('Ford', 'Chevy', 'Dodge');
sort($cars);

print_r($cars);
```

```
Array
(
    [0] => Chevy
    [1] => Dodge
    [2] => Ford
)
```

Passing By Reference

- Functions receives reference to value rather than a copy of the value.
- Function can modify value and will be seen by it's caller.
- Provides additional channel of communication between function and caller.
- Makes it more difficult to track effects of function call and harder to track down bugs.

But...WHY?

Passing By Reference

- Since the value is not copied, the memory footprint is lower. Function *doesn't have to* modify contents.
- Returning multiple values.

```php
function isAgeValid($age, &$messages)
{
    if(is_numeric($age) && $age > 0 && $age < 122) {
        return true;
    }
    $messages[] = 'Age is Invalid';
    return false;
}

function isNameValid($name, &$messages)
{
    $match = preg_match('/^[a-z]{3,20}$/i', $name);
    if(is_numeric($match) && 0 < $match) {
        return true;
    }
    $messages[] = 'Name is Invalid';
    return false;
}

$messages  = array();
$ageValid  = isAgeValid(231, $messages);
$nameValid = isNameValid('a1b2', $messages);

var_dump($ageValid);
var_dump($nameValid);
print_r($messages);
```

```
bool(false)

bool(false)

Array
(
    [0] => Age is Invalid
    [1] => Name is Invalid
)
```

Review Anonymous Functions

Anonymous Functions

- New to PHP 5.3.
- Functions with no name.
- Can be assigned to variables.
- Customized code w/in a function at the time it's created = flexibility.
- Short-term disposable functions: usually if another function takes a "callback".

Callbacks

- A callback is an answer to a question.
- It's used by a function doing something generic, to perform a more specific and customized task.

Callback Example

array_map (*callable* $callback , *array* $arr1)

- Looping over array is done by array_map.
- Looping over array is the generic behavior.
- *callable* allows you to add customization.

Callback Example

array_map (*callable* $callback , *array* $arr1)

- For each element in the array, do something.
- What to do is defined by the callable, aka anonymous, function.
- Example: We want to convert integers to the string value in the array, 1 should become one, etc... Define a function to do it.

Callback Example

```php
$conversion = function($integer) {
    $map = array(
        1 => 'one',
        2 => 'two',
        3 => 'three',
        4 => 'four',
        5 => 'five',
        // etc...
    );
    return (isset($map[$integer]))
        ? $map[$integer]
        : false;
};

$integers = array(2, 3, 4);
$strings  = array_map($conversion, $integers);

print_r($integers);
print_r($strings);
```

```php
$conversion = function($integer) {
    $map = array(
        2 => 'two',
        3 => 'three',
        4 => 'four',
        // etc...
    );
    return (isset($map[$integer]))
        ? $map[$integer]
        : false;
};

function customArrayMap(Closure $function, array $array)
{
    $updatedArray = array();
    foreach($array as $key => $value) {
        $updatedArray[$key] = $function($value);
    }
    return $updatedArray;
}

$integers = array(2, 3, 4);
$strings  = array_map($conversion, $integers);

print_r($integers);
print_r($strings);
```

Recursion

- A function that calls itself
- Must have some end condition
- If end condition is not met, will call itself forever
- The end condition must eventually be met
- Examples: calculating fibonacci and factorials, iterating over an array with an unknown number of nested arrays.

Recursion and Factorials

$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Factorials with a *for* loop

```php
$num    = 1;
$factor = 6;

for ($x = $factor; $x > 0; $x--) {
    $num = $num * $x;
}

echo $num; // prints: 720
```

Recursion and Factorials

```php
<?php

function factorial($num) {
    return ($num == 0) ? 1 : $num * factorial($num - 1);
}

echo factorial(6);
```

Recursion

```php
$nav = array(
    'Finance' => array(
        'Markets'     => 'markets.php',
        'Your Money' => 'money.php',
    ),
    'Politics' => array(
        'Defense'      => 'defense.php',
        'Politicians' => array(
            'Obama' => 'obama.php',
            'Democrats' => array(
                'Obama'    => 'obama.php',
                'Clinton' => 'clinton.php',
            ),
        ),
    ),
);
```

Recursion

Solution w/out recursion: nested conditionals with nested foreach loops.

Problem: Must know how many levels deep to go.

```
foreach ($nav as $key => $val) {
    if (is_array($val)) {
        foreach ($val as $key2 => $val2) {
            // etc...
        }
    }
}
```

Recursion

Solution with recursion: no need to predict or know how many levels to go.

```php
function nav(array $data)
{
    $nav = "<ul>";
    foreach($data as $key => $val) {
        if(is_array($val)) {
            $nav .= "<li>$key" . nav($val) . "</li>";
        } else {
            $nav .= "<li><a href='$val'>$key</a></li>";
        }
    }
    return $nav . "</ul>";
}

echo nav($nav);
```

Recursion

Solution with recursion: no need to predict or know how many levels to go.

- Finance
  - Markets
  - Your Money
- Politics
  - Defense
  - Politicians
    - Obama
    - Democrats
      - Obama
      - Clinton

Server Side Validation

Server Side Validation

- Validation done by software (PHP) running on the server.
- Ensure that data that was submitted by a user is valid and what you expect.
- Must be done regardless of whether our client side (javascript) validation runs.
- Example: A form is submitted with empty or invalid values: it's pointless to store this in the database, or send an email, or do anything else with this data.

Server Side vs. Client Side Validation

- Client side validation is typically done with JavaScript.
- It's tempting to implement because you can notify the user much faster since the form doesn't have to be submitted first.
- It should be thought of as an improvement to the user's experience, however, not a replacement for server side validation.

Why both or only server side validation?

- Client side validation can be disabled, browsers have ways of disabling JavaScript.
- If JavaScript is disabled, or html forms are modified, how does the data get validated?
- What if user doesn't use the browser (curl, PHP, other scripting language, etc...) to submit data to the server?

Server Side Validation

Only *some* built in PHP functions...

- isset()
- empty()
- is_string()
- is_numeric()
- is_int()
- is_array()
- strlen()
- array_key_exists()
- ctype_alnum()
- ctype_alpha()
- ctype_lower()
- ctype_upper()

Server Side Validation

Usage examples...

- is_string() - usernames, people names, cities
- is_numeric() - age, years, decimal values
- is_int() - age, years, but cannot be string values
- ctype_alnum() - usernames, passwords
- ctype_alpha() - first and last names, cannot have spaces

isset() vs. empty()

```php
var_dump( isset($something) ); // bool(false)
var_dump( empty($something) ); // bool(true)

$something = '';
var_dump( isset($something) ); // bool(true)
var_dump( empty($something) ); // bool(true)

$something = 0;
var_dump( isset($something) ); // bool(true)
var_dump( empty($something) ); // bool(true)

$something = array();
var_dump( isset($something) ); // bool(true)
var_dump( empty($something) ); // bool(true)

$something = 'hi, there.';
var_dump( isset($something) ); // bool(true)
var_dump( empty($something) ); // bool(false)
```

isset() vs. empty()

isset()

- false if variable has never been declared and initialized.
- true if variable is false, which means (int) 0, (float) 0.0, an empty array(), (bool) false, etc...
- true in all other situations.

empty()

- true if variable has never been declared and initialized.
- true if variable is false, which means (int) 0, (float) 0.0, an empty array(), (bool) false, etc...
- false in all other situations.

```php
$messages = array();

if(strtolower($_SERVER['REQUEST_METHOD']) === 'post') {
    if (empty($_POST['username']) || !ctype_alnum($_POST['username'])) {
        $messages['username'] = 'Username is invalid. Must be alpha-numeric';
    }

    if (empty($_POST['first_name']) || !ctype_alpha($_POST['first_name'])) {
        $messages['first_name'] = 'First Name is invalid. Must be alpha';
    }

    if (empty($_POST['age']) || !is_numeric($_POST['age'])) {
        $messages['age'] = 'Age is invalid. Must be numeric';
    }
}
?>
<form action="<?= $_SERVER['PHP_SELF'] ?>" method="post">
    <? if(array_key_exists('username', $messages)) { ?>
    <p><? echo $messages['username']; ?></p>
    <? } ?>
    <input type="text"   name="username" />

    <? if(array_key_exists('first_name', $messages)) { ?>
    <p><? echo $messages['first_name']; ?></p>
    <? } ?>
    <input type="text"   name="first_name" />

    <? if(array_key_exists('age', $messages)) { ?>
    <p><? echo $messages['age']; ?></p>
    <? } ?>
    <input type="text"   name="age" />
    <input type="submit" value="Submit" />
</form>
```

Checking Arrays

- isset() and array_key_exists() are very similar but not quite interchangeable.
- isset() returns *true* for array keys that are NULL, whereas array_key_exists() returns *false*.
- isset() is much faster... see next slide.

Checking Arrays

isset() is "much faster" is a relatively misleading statement... But it's important to know.

100,000 iterations:

array_key_exists() : 205 ms
isset() : 35ms

Checking Arrays

```php
$data = array(
    'name'       => 'New York',
    'population' => null,
);

var_dump ( isset($data['population']) );            // bool (false)
var_dump ( array_key_exists('population', $data) ); // bool (true)

var_dump ( isset($data['size']) );                  // bool (false)
var_dump ( array_key_exists('size', $data) );       // bool (false)
```

True
False

```php
$data = array(
    'name'        => 'new york',
    'population' => 8391881,
    'boroughs'    => array(
        'manhattan' => array(
            'population' => 1585873,
        ),
        'brooklyn'  => array(
            'population' => 2504700,
        ),
    ),
);

// find the name
echo $data['name'];
// find the population of brooklyn
echo $data['boroughs']['brooklyn']['population'];

// WE DON'T HAVE A FULL DATASET, WHAT ABOUT THE BRONX?

// find the population of the Bronx
echo $data['boroughs']['bronx']['population'];
```

Notice: Undefined index: bronx in /Applications/MAMP/htdocs/slides2.php on line 24

```php
$data = array(
    'name'        => 'new york',
    'population' => 8391881,
    'boroughs'    => array(
        'manhattan' => array(
            'population' => 1585873,
        ),
        'brooklyn'  => array(
            'population' => 2504700,
        ),
    ),
);

// find the name
echo $data['name'];
// find the population of brooklyn
echo $data['boroughs']['brooklyn']['population'];

// WE DON'T HAVE A FULL DATASET, WHAT ABOUT THE BRONX?

// find the population of the Bronx
echo $data['boroughs']['bronx']['population'];
```

FAIL!

Checking Arrays

```php
// MUST do this...
if ( array_key_exists('bronx', $data['boroughs']) &&
     array_key_exists('population', $data['boroughs']['bronx'])
) {
    echo $data['boroughs']['bronx']['population'];
}

// CANNOT do this...
if ( array_key_exists('population', $data['boroughs']['bronx']) ) {
    echo $data['boroughs']['bronx']['population'];
}
```

When to Check Arrays

- When you cannot guarantee the structure because it's coming from somewhere else. For example, a form. Or if it's being passed into a user-defined function.
- If you know that the array keys will exist, don't bother checking. But you **must** know. For example, if you declared the array yourself, or if it's coming from a database where the structure is always guaranteed.

Escaping Input

- User inputted data *cannot* and *should never* be trusted.
- May have unwanted HTML tags.
- May have malicious JavaScript code.
- May have malicious SQL statements.

Escaping Input

Scenario 1: User inputs some HTML along with their first name: Dave. If we don't "escape" this data, when we display their name to the browser it will be bold!

Escaping Input

Scenario 2: User inputs some JavaScript along with the comment they just posted. Since the comment is public, the JavaScript will now be executed on every page request for every user.

The JavaScript may read other users' Cookie data, and send it to the malicious user's server. This data can be used for *session hijacking* and the malicious user can now log in as anyone who visits the page with their comment.

Escaping Input is Essential!

So... How?

Escaping Input

Built in PHP functions...

- strip_tags()
- htmlentities()
- html_entity_decode()
- mysql_real_escape_string()

strip_tags()

```php
$data  = '<strong>Hello, There!</strong>';
$data .= '<iframe src="http://malicious.com" />';

echo $data;
echo strip_tags($data);
echo strip_tags($data, '<strong><b>');
```

```
<strong>Hello, There!</strong><iframe src="http://malicious.com" />
```

```
Hello, There!
```

```
<strong>Hello, There!</strong>
```

htmlentities()

- Converts special characters like < and > into html entities.
- An example of an html entity is: <
- This special combination of characters tells the browser to render the < sign rather than interpret it as actual HTML.
- There are hundreds of special characters like this, such as the copyright sign and the registered trademark sign.

htmlentities()

- & (ampersand) becomes &
- " (double quote) becomes " when ENT_NOQUOTES is not set (default).
- ' (single quote) becomes ' only when ENT_QUOTES is set.
- < (less than) becomes <
- > (greater than) becomes >

Why use htmlentities()

- If we have strip_tags, why not just use that?
- Perhaps, you want to preserve the exact representation of what the user submitted.
- Perhaps you want to display exactly what the user submitted.
- Perhaps you want to be able to store the HTML entity version, but convert HTML entities back to actual HTML (WYSIWYG editor like TinyMCE).

```php
$data  = "<strong>Hello, There!</strong>";
$data .= '<iframe src="http://malicious.com" />';

echo $data;

$data = htmlentities($data);
echo $data;

$data = html_entity_decode($data);
echo $data;
```

```
<strong>Hello, There!</strong>
<iframe src="http://malicious.com" />


&lt;strong&gt;Hello, There!&lt;/strong&gt;
&lt;iframe src=&quot;http://malicious.com&quot; /&gt;


<strong>Hello, There!</strong>
<iframe src="http://malicious.com" />
```

Date Validation

Basic Date Validation

Two built-in functions to help:

- checkdate(*int* $month, *int* $day, *int* $year)
- strtotime(*string* $time)

checkdate()

checkdate(*int* $month, *int* $day, *int* $year)

- This function is useful if you have the day, month, and year all as separate values.
- Most of the time this is not the case.
- If you have a form where the user fills out each the day, month, and year as separate values, use this function.

checkdate()

checkdate(*int* $month, *int* $day, *int* $year)

```php
var_dump( checkdate(12, 24, 2005) ); // bool (true)
var_dump( checkdate(1, 6, 1983) );   // bool (true)
var_dump( checkdate(2, 29, 2011) );  // bool (false)
var_dump( checkdate(2, 29, 2012) );  // bool (true)
```

strtotime()

strtotime(*string* $time)

The argument can be virtually any date string

- "now"
- "10 September 2000"
- "+1 day"
- "+1 week"
- "+1 week 2 days 4 hours 2 seconds"
- "next Thursday"
- "last Monday"

strtotime()

strtotime(*string* $time)

The function returns *false* if it cannot parse the string, in other words, if the date is invalid.

If the date is valid, however, it returns a Unix timestamp.

The unix timestamp is the number of seconds since the Unix Epoch, December 31, 1969 23:59:60.

Unix timestamps are used all over... built-in PHP date functions, database applications, other programming languages, etc...

strtotime()

```php
// Valid dates.
var_dump( strtotime("now") );                                  // int(1334811838)
var_dump( strtotime("10 September 2000") );                    // int(968558400)
var_dump( strtotime("+1 day") );                               // int(1334898238)
var_dump( strtotime("+1 week") );                              // int(1335416638)
var_dump( strtotime("+1 week 2 days 4 hours 2 seconds") );     // int(1335603840)
var_dump( strtotime("next Thursday") );                        // int(1335412800)
var_dump( strtotime("last Monday") );                          // int(1334548800)

// Invalid dates.
var_dump( strtotime("invalid date") );                         // bool(false)
var_dump( strtotime("January 42, 2012") );                     // bool(false)

// Example usage.
if (false === strtotime($_POST['birthdate'])) {
    echo "Your birth date is invalid.";
}
```

strtotime()

Using the date() function to format the timestamp:

```php
echo date( 'Y-m-d', strtotime("+1 week") ); // 2012-04-24
echo date( 'F', strtotime("last month") );  // March
```

Regular Expressions

Regular Expressions

- Used for matching patterns in strings.
- Can be used to match and then replace a pattern in a string.
- Amazingly powerful.
- Extremely scary to developers.

Regular Expressions

Some examples:

```
// Matches a username
/^[a-z0-9_-]{3,16}$/

// Matches an email address
/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
```

WAT?

Regular Expressions

- Read the chapter on Regular Expressions in the book.
- Next week we will spend 1 hour going over how to create simple regular expressions.