WEBD 236

Web Information Systems Programming

Week 8

(todd.whittaker@franklin.edu)

FRANKLIN UNIVERSITY

Agenda

- This week's expected outcomes
- This week's topics
- This week's homework
- Upcoming deadlines
- Solution to Homework 6
- Solution to Lab 2
- Questions and answers

FRANKLIN UNIVERSITY

Week 8 Outcomes

- Write regular expressions that test, capture, and replace data within strings
- Explain the purpose and use of exception handling for error detection and correction
- Use the keywords try, throw, and catch to implement exception handling
- Use regular expressions and exception handling to validate data.

Regular Expressions

- Regular expressions: a language of its own
  - Idea: data is often in a structured format e.g.
    - E-mail addresses
    - URLs
    - Phone numbers
    - Dates/times
  - We want to be able to ensure that data conforms to the expected format and extract it in a usable way.

Regular expressions

- Even a phone number can be written many ways:
  - 614-947-6110
  - (614) 947-6110
  - 614.947.6110
- What we want in the DB:
  - "6149476110"

Using string functions and PHP code to recognize each of these as valid and extracting them for storage is difficult.

Regular Expressions

- RegEx is a language for describing patterns
  - A phone number consists of:
    - An optional "("
    - Three digits
    - An optional ")"
    - An optional separator `[. -]`
    - Three digits
    - An optional separator `[. -]`
    - Four digits

Regular Expressions

- RegEx is a language for describing patterns
  - A phone number consists of:
    - An optiona[illegible]
    - Thre[illegible]
    - An o[illegible]
    - An optional separator `[. -]`
    - Three digits
    - An optional separator `[. -]`
    - Four digits

```
/^\(?\d{3}\)?[. -]?\d{3}[. -]?\d{4}$/
```


Regular Expressions

- Two pieces of data needed
  - The pattern: a string that contains a syntactically correct regular expression.
  - The subject: a string that you will match against the pattern.
- A function to do the matching
  - `preg_match($pattern, $subject)`

Regular Expressions

- Example

```
$pattern = '/^\(?\d{3}\)?[. -]?\d{3}[. -]?\d{4}$/';
$subject = '614-947-6110';
preg_match($pattern, $subject); // returns true
```


Regular Expressions

- Building patterns
  - Enclose patterns in slashes: e.g. `'/stuff/'`
  - Most characters represent themselves
  - Some characters need to be escaped

Pattern	Matches
`\\`	Backslash character
`\/`	Forward slash character
`\t \n \r \f`	Tab, newline, carriage return, form feed
`\xhh`	Any Latin-1 character whose ord is the hex number hh.

Regular Expressions

- Building patterns
  - Some sequences represent a whole class of characters at once

Pattern	Matches
`.`	Any single character
`\w`	Any single letter, number, or underscore
`\W`	Any single character not a letter, number, or underscore
`\d`	Any single digit (0 through 9)
`\D`	Any single character not a digit
`\s`	Any single whitespace character (space, tab, newline, etc)
`\S`	Any single character that is not whitespace

Regular Expressions

- Building patterns
  - You can define your own character classes by enclosing the characters in [ ].
    - Examples
      - `'[aeiouy]'` matches any single vowel
      - `'[a-z]'` matches any lower case letter
      - `'[a-zA-Z]'` matches any single letter
      - `'[^aeiouy]'` matches any single non-vowel
      - `'[a-zA-Z0-9_]'` is the same as `'\w'`

Regular Expressions

- Building patterns
  - You can define your own character classes by enclosing the characters in [ ].
    - Examples
      - '[aeiouy]' matches any single vowel
      - '[a-z]' matches any lower case letter
      - '[a-zA-Z]' matches any single letter
      - '[^aeiouy]' matches any single non-vowel
      - '[a-zA-Z0-9_]' the same as '\w'

The caret (^) is the not operator in a character class.

Regular Expressions

- Building patterns
  - Common useful character classes are predefined

Pattern	Matches
[:digit:]	All digits (i.e. '\d' or '[0-9]')
[:lower:]	All lower case characters (i.e. '[a-z]')
[:upper:]	All upper upper case characters (i.e. '[A-Z]')
[:letter:]	All letters (i.e. '[a-zA-Z]')
[:alnum:]	All letters or digits (i.e. '[a-zA-Z0-9]')
[:word:]	All letters, digits, or underscore (i.e. '[a-zA-Z0-9_]')
[:print:]	All printable characters including space
[:graph:]	All printable characters excluding space
[:punct:]	All printable characters excluding letters and digits

Regular Expressions

- Building patterns
  - Matching a single character is less useful than matching repeating groups of characters.

Pattern	Matches
*	Matches 0 or more of the previous pattern
+	Matches 1 or more of the previous pattern
?	Matches 0 or 1 of the previous pattern
{n}	Matches exactly *n* of the previous pattern
{n,}	Matches *n* or more of the previous pattern
{n,m}	Matches between *n* and *m* of previous pattern.

Regular Expressions

- Building patterns
  - Logical operators

Pattern	Matches
^	In a character class, means not
|	Matches left or right hand side
ab	Matches a (any character) followed by b (logical and)

  - Subgroups, beginning and end of line

Pattern	Matches
^	Matches the beginning of a line
$	Matches the end of line
(pattern)	Creates a subgroup matching pattern

Regular Expressions

- Back to the phone number example

```
$pattern = '/^\(?\d{3}\)?[. -]?\d{3}[. -]?\d{4}$/';
$subject = '614-947-6110';
preg_match($pattern, $subject); // returns true
```

  - ^ – match beginning of line
  - \(? – match 0 or 1 (characters
  - \d{3} – match three digits
  - \)? – match 0 or 1) characters

Regular Expressions

- Back to the phone number example

```
$pattern = '/^\(?\d{3}\)?[. -]?\d{3}[. -]?\d{4}$/';
$subject = '614-947-6110';
preg_match($pattern, $subject); // returns true
```

  - [. -]? – match 0 or 1 dots, spaces, dashes
  - \d{3} – match three digits
  - [. -]? – match 0 or 1 dots, spaces, dashes
  - \d{4} – match four digits
  - $ – match end of line

Regular Expressions

- More examples
  - Money amounts
    - Match a string like “$4321.52,” “4321.52,” “4321,” “$4321,” and “$0.02” but not “$.02” or “0.5”

Regular Expressions

- More examples
  - Money amounts
    - Match a string like “$4321.52,” “4321.52,” “4321,” “$4321,” and “$0.02” but not “$.02” or “0.5”

```
$data = array('$4321.52', '4321.52', '4321', '$4321', '$0.02',
'$.02', '0.5', 'abc');
$pattern = '/^\$?\d+([.]?\d{2})?$/';
foreach ($data as $datum) {
    if (preg_match($pattern, $datum)) {
        print "$pattern matches $datum<br />";
    } else {
        print "$pattern does not match $datum<br />";
    }
}
```

Regular Expressions

- More examples
  - File names of pictures
    - Match a string like “foo.jpg,” “bar.png,” “baz.GIF,” but not “file.docx” or “file.pdf”

Regular Expressions

- More examples
  - File names of pictures
    - Match a string like “foo.jpg,” “bar.png,” “baz.GIF,” but not “file.docx” or “file.pdf”

```
$data = array('foo.jpg', 'bar.png', 'B-A-Z.GIF', 'file.docx');
$bad = '<>:"\/\\\\|?*"';
$ptrn = "/^[^$bad]+[.](jpg|jpeg|gif|png|tif|tiff|bmp|svg)$/i";
foreach ($data as $datum) {
    if (preg_match($ptrn, $datum)) {
        print "$ptrn matches $datum<br />";
    } else {
        print "$ptrn does not match $datum<br />";
    }
}
```

Regular Expressions

- More examples
  - HTML tags
    - Match a string like “,” “,” “<a href=‘foo.php’>, ”but not “1 < 2” or “:>)”

Regular Expressions

- More examples
  - HTML tags
    - Match a string like “,” “,” “<a href=‘foo.php’>, ”but not “1 < 2” or “:>)”

```
$data = array('<i>', '</i>', '<a href="foo.php">', '1 < 2');
$pattern = '/^<\/?[a-zA-Z]+[^>]*>$/';
foreach ($data as $datum) {
    $escaped = htmlentities($datum);
    if (preg_match($pattern, $datum)) {
        print "$pattern matches $escaped<br />";
    } else {
        print "$pattern does not match $escaped<br />";
    }
}
```

Regular Expressions

- Other regular expression functions
  - `preg_replace` – Replaces each matched occurrence with a different string.
    - Subgroups within `(  )` are *captured*
    - Subgroups within `(:?  )` are not captured
    - A captured subgroup can appear in the replacement string as `$1`, `$2`, `$3`, etc.

Regular Expressions

- Other regular expression functions
  - `preg_replace` – Replaces each matched occurrence with a different string.

```
$data = array('614-947-6110', '(614) 947-6110', '614.947.6110');
$pattern = '/^\(?(\d{3})\)?[. -]?(\d{3})[. -]?(\d{4})$/';
foreach ($data as $datum) {
    $result = preg_replace($pattern, "$1$2$3", $datum);
    print("$datum became $result<br />");
}
```

Regular Expressions

- Other regular expression functions
  - `preg_match_all` – Finds all matches and returns a count. Fills an array with matches when given as a parameter.
    - Ex: splitting a string into tags using spaces & commas

```
$data = ',abc  def,  ghi,, jkl,';
$pattern = '/[^ ,]+/';
$matches = array();
$result = preg_match_all($pattern, $data, $matches);
print_r($matches[0]);
```


Regular Expressions

- Other regular expression functions
  - `preg_match_all` – Finds all matches and returns a count. Fills an arra…hen given as a parameter.
    - Ex: splitting a string into tags u…has

```
Array
(
    [0] => abc
    [1] => def
    [2] => ghi
    [3] => jkl
)
```

```
$data = ',abc  def,  ghi,, jkl,';
$pattern = '/[^ ,]+/';
$matches = array();
$result = preg_match_all($pattern, $data, $matches);
print_r($matches[0]);
```

Regular Expressions

- Other regular expression functions
  - `preg_split` – splits a string into an array of strings using a regular expression as a delimiter.
    - Ex: splitting a string into tags using spaces & commas

```
function nonempty($val) {
    return $val;
}
$data = ',abc  def,  ghi,, jkl,';
$pattern = '/[ ,]+/';
$result = array_filter(preg_split($pattern, $data), 'nonempty');
print_r($result);
```


Regular Expressions

- Other regular expression functions
  - `preg_split` – splits a string into an array of strings using a regular expres…er.
    - Ex: splitting a string into tags u…has

```
Array
(
    [1] => abc
    [2] => def
    [3] => ghi
    [4] => jkl
)
```

```
function nonempty($val) {
    return $val;
}
$data = ',abc  def,  ghi,, jkl,';
$pattern = '/[ ,]+/';
$result = array_filter(preg_split($pattern, $data), 'nonempty');
print_r($result);
```

Regular Expressions

- Miscellaneous
  - Regular expressions are easier to write than read
  - Build them incrementally, testing often
  - Use a web service to help you build/test (http://www.solmetra.com/scripts/regex/index.php)
  - Some things look deceptively simple but are actually very complex (i.e. e-mail addresses)
  - If you like regular expressions, you'll love Perl.

Exception Handling

- What is an exception?
  - A method of reporting error conditions, e.g.
    - When the database connection fails
    - When the disk fills while writing a file
    - When the network fails while sending/receiving data
  - Exceptions alter the flow of control of a program
    - Current execution stops.
    - The closest *exception handler* begins executing.
    - If there is no handler, the program halts.

Exception Handling

- Throwing an exception

```
print "This will appear";
throw new Exception("Something bad happened.");
print "This will not appear";
```

```
This will appear

Fatal error: Uncaught exception 'Exception' with message
'Something bad happened.' in
C:\xampp\htdocs\exceptions\exceptions.php:3
Stack trace:
#0 {main}
  thrown in C:\xampp\htdocs\exceptions\exceptions.php on line 3
```


Exception Handling

- Throwing an exception

```
print "This will appear";
throw new Exception("Something bad happened.");
print "This will not appear";
```

```
This will appear

Fatal error: Uncaught exception
'Something bad happened.' in
C:\xampp\htdocs\exceptions\exceptions.php:3
Stack trace:
#0 {main}
  thrown in C:\xampp\htdocs\exceptions\exceptions.php on line 3
```

Throw an `Exception` object with an error message parameter to the constructor.

Exception Handling

- Throwing an exception

```
print "This will appear";
throw new Exception("Something bad happened.");
print "This will not appear";
```

```
This will appear

Fatal error: Uncaught exce                    age
'Something bad happened.'
C:\xampp\htdocs\exceptio
Stack trace:
#0 {main}
  thrown in C:\xampp\htdocs\exceptions\exceptions.php on line 3
```

The stack trace tells you what functions were called in sequence leading to the error.

Exception Handling

- Why throw exceptions?
  - The place at which you detect an error and the place at which you can correct the error are often different.
  - Exceptions let you alter the execution path to get back to the place where you can correct the problem.

Exception Handling

- Catching exceptions
  - Sequence:
    - "try" to execute code that may generate an exception
    - "catch" any exceptions that are thrown
  - We have seen this before:

```
global $db;
try {
    $db = new PDO('sqlite:somedatabase.db3');
} catch (PDOException $e) {
    die("Could not open database. " . $e->getMessage());
}
```


Exception Handling

- Catching exceptions
  - Sequence:
    - "try" t… te an exception
    - "catch" …
  - We have…

How can we know that the PDO constructor may throw a PDOException?

```
global $db;
try {
    $db = new PDO('sqlite:somedatabase.db3');
} catch (PDOException $e) {
    die("Could not open database. " . $e->getMessage());
}
```

Exception Handling

- Catching exceptions
  - Several methods of every exception:

Method	Description
`getMessage()`	Error message given to constructor
`getCode()`	Error code given to constructor
`getFile()`	Name of the file where exception was thrown
`getLine()`	Line number in file where exception was thrown
`getTrace()`	Array of function calls leading to exception
`getTraceAsString()`	String representation of the trace

Exception Handling

- You can create your own exception classes

```
class MyException extends Exception {
    public function __construct($message) {
        parent::__construct($message);
    }
}
```

This is useful because there's useful information in the *type* of the exception (e.g. `ValidationException` vs. `PDOException`).

Exception Handling

- You can use the type of the exception to change what you do when you catch.

```
try {
    doSomething(5);
} catch (MyException $e) {
    // do something to recover here
} catch (Exception $e) {
    die("Caught " . get_class($e) . ': ' . $e->getMessage());
}
```


Exception Handling

- A more reasonable example of exceptions

```
class User extends Model {
    protected $phoneNumber;
    // ... more properties here...
    public function setPhoneNumber($num) {
       $p = '/^\s*\(?(\d{3})\)?[. -]?(\d{3})[. -]?(\d{4})\s*$/';
       if (!preg_match($p, $num)) {
           throw new InvalidArgumentException(
               "Expected a phone number, got $num");
        }
        $this->phoneNumber = preg_replace($p, "$1$2$3", $num);
        return $this;
    }
    // ... more methods here...
}
```

Exception Handling

- A more reasonable example of exceptions

```
class User extends Model
    protected $phoneNumbe
    // ... more propertie
    public function setPho
       $p = '/^\s*\(?(\d{3})\)
       if (!preg_match($p, $nu
           throw new InvalidArgumentException(
               "Expected a phone number, got $num");
        }
        $this->phoneNumber = preg_replace($p, "$1$2$3", $num);
        return $this;
    }
    // ... more methods here...
}
```

Where did `InvalidArgumentException` come from?

Exception Handling

- Some built-in exception types
  - `InvalidArgumentException`
  - `LengthException`
  - `LogicException`
  - `OutOfBoundsException`
  - `OutOfRangeException`
  - `RuntimeException`
  - `UnexpectedValueException`

Exception Handling

- Some built-in exception types
  - `InvalidArgumentException`
  - `LengthExcept`
  - `LogicExc`
  - `OutOfBou`
  - `OutOfRan`
  - `RuntimeException`
  - `UnexpectedValueException`

These all exist in the Standard PHP Library, documentation at http://www.php.net/manual/en/spl.exceptions.php

A Simple Validator

- A simpler validator class than the book
  - Design goals
    - Regex-based pattern matching
    - Methods for validating common data (email, integer, float, ranges, non-empty, money, etc.)
    - Accrues error messages for easy rendering

A Simple Validator

- A simpler validator class than the book

```
class Validator {
    private $errors;

    public function __construct() {
        $this -> errors = array();
    }
    public function hasErrors() {
        return count($this -> errors) > 0;
    }
    public function allErrors() {
        return $this -> errors;
    }
    private function addError($key, $message) {
        $this -> errors[$key] = $message;
    }
```

A Simple Validator

- A simpler validator class than the book

```
    public function errorsFor($key) {
        if (isset($this -> errors[$key])) {
            return $this -> errors[$key];
        }
        return '';
    }
    public function required($key, $value, $message = false) {
        $pattern = '/[[:graph:]]+/';
        $message = $message ? $message : "Field is required";
        if (!preg_match($pattern, $value)) {
            $this -> addError($key, $message);
            return false;
        }
        return true;
    }
```

A Simple Validator

- A simpler validator class than the book

```
public function float($key, $value, $message = false) {
    $pattern = '/^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$/';
    $message = $message ? $message : "Not a valid float";
    if (!preg_match($pattern, $value)) {
        $this -> addError($key, $message);
        return false;
    }
    return true;
}
```

A Simple Validator

- A simpler validator class than the book

```
public function password($key, $value, $message = false) {
    $message = $message ? $message : "Not strong enough.";
    $patterns = array(
        '/^[[:graph:]]{8,}$/', # all printable, 8 in length
        '/[[:upper:]]/',       # at least 1 upper
        '/[[:digit:]]/',       # at least 1 digit
        '/[[:punct:]]/');      # at least 1 symbol
    foreach ($patterns as $pattern) {
        if (!preg_match($pattern, $value)) {
            $this -> addError($key, $message);
            return false;
        }
    }
    return true;
}
```

Using the Validator

- In our ToDo application…

```
function post_update($params) {
    ensureLoggedIn();

    $id = trim(safeParam($_POST, 'id'));
    $description = trim(safeParam($_POST, 'description'));
    $done = trim(safeParam($_POST, 'done'));

    $validator = new Validator();
    $validator -> required('id', $id, 'No ID specified');
    $validator -> required('description', $description,
        'Description required');
    $validator -> required('done', $done, "Done is required");

    // continued ...
```

Using the Validator

- In our ToDo application…

```
    if (!$validator -> hasErrors()) {
        $todo = Todo::findById($id);
        if ($todo) {
            $todo -> setDescription($description);
            $todo -> setDone($done);
            $todo -> update();
        }
        redirectRelative("todo/view/$id");
    }

    // continued ...
```

Using the Validator

- In our ToDo application…

```
    $todo = new Todo($id, $description, $done);
    renderTemplate(
        "views/todo_edit.inc",
        array(
            'title' => 'Editing To Do',
            'errors' => $validator -> allErrors(),
            'todo' => $todo
        )
    );
}
```

Using the Validator

- In our ToDo application…
  - Full source code available at http://cs.franklin.edu/~whittakt/WEBD236

Upcoming Deadlines

- Readings for next week
  - Chapters 16 and 17 in *PHP and MySQL*
- Assignments
  - Midterm exam due February 26
  - Homework 7 due March 4
  - Lab 3 due March 4
- Next week:
  - Database design, database creation with SQL

Solution to HW 6

Solution to Lab 2

General Q & A

- Questions?
- Comments?
- Concerns?