CSE 154

LECTURE 19: REGULAR EXPRESSIONS

Uploading details

```html
<input type="file" name="avatar" />
```
HTML

| Browse… No file selected. | Submit Query |
output

- example: if you upload borat.jpg as a parameter named avatar,

  - $_FILES["avatar"]["name"] will be "borat.jpg"
  - $_FILES["avatar"]["type"] will be "image/jpeg"
  - $_FILES["avatar"]["tmp_name"] will be something like "/var/tmp/phpZtR4TI"

Processing uploaded file, example

```php
$username = $_POST["username"];
if (is_uploaded_file($_FILES["avatar"]["tmp_name"])) {
    move_uploaded_file($_FILES["avatar"]["tmp_name"],
    "$username/avatar.jpg");
  print "Saved uploaded file as $username/avatar.jpg\n";
} else {
  print "Error: required file not uploaded";
}                                                      PHP
```

- functions for dealing with uploaded files:
  - is_uploaded_file(filename)
  - returns TRUE if the given filename was uploaded by the user
  - move_uploaded_file(from, to)
  - moves from a temporary file location to a more permanent file
- proper idiom: check is_uploaded_file, then do move_uploaded_file

What is form validation?

- **validation**: ensuring that form's values are correct

- some types of validation:
  - preventing blank values (email address)
  - ensuring the type of values
    - integer, real number, currency, phone number, Social Security number, postal address, email address, date, credit card number, …
  - ensuring the format and range of values (ZIP code must be a 5-digit integer)
  - ensuring that values fit together (user types email twice, and the two must match)

A real form that uses validation

Client vs. server-side validation

Validation can be performed:

- **client-side** (before the form is submitted)
  - can lead to a better user experience, but not secure (why not?)

- **server-side** (in PHP code, after the form is submitted)
  - needed for truly secure validation, but slower

- both
  - best mix of convenience and security, but requires most effort to program

An example form to be validated

```html
<form action="http://foo.com/foo.php" method="get">
  <div>
    City:   <input name="city" /> <br />
    State: <input name="state" size="2" maxlength="2" /> <br />
    ZIP:    <input name="zip" size="5" maxlength="5" /> <br />
    <input type="submit" />
  </div>
</form>
```
<div align="right">HTML</div>

City: [_____]
State: [__]
ZIP: [____]
[Submit Query]

<div align="right">output</div>

- Let's validate this form's data on the server...

Basic server-side validation

```php
$city  = $_POST["city"];
$state = $_POST["state"];
$zip   = $_POST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) != 5) {
  print "Error, invalid city/state/zip submitted.";
}                                                        PHP
```

- *basic idea:* examine parameter values, and if they are bad, show an error message and abort. But:

  - How do you test for integers vs. real numbers vs. strings?

  - How do you test for a valid credit card number?

  - How do you test that a person's name has a middle initial?

  - (How do you test whether a given string matches a particular complex format?)

Regular expressions

```
/^[a-zA-Z_\-]+@(([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4}$/
```

- **regular expression** ("regex"): a description of a pattern of text
  - can test whether a string matches the expression's pattern
  - can use a regex to search/replace characters in a string
- regular expressions are extremely powerful but tough to read
  (the above regular expression matches email addresses)
- regular expressions occur in many places:
  - Java: Scanner, String's split method (CSE 143 sentence generator)
  - supported by PHP, JavaScript, and other languages
  - many text editors (TextPad) allow regexes in search/replace
  - The site Rubular is useful for testing a regex.

Regular expressions

This picture best describes regex.

Basic regular expressions

```
/abc/
```

- in PHP, regexes are strings that begin and end with /

- the simplest regexes simply match a particular substring

- the above regular expression matches any string containing "abc":

  - YES: "abc", "abcdef", "defabc", ".=.abc.=.", ...

  - NO: "fedcba", "ab c", "PHP", ...

Wildcards: .

- A dot . matches any character except a \n line break
  - /.oo.y/ matches "Doocy", "goofy", "LooNy", ...
- A trailing i at the end of a regex (after the closing /) signifies a case-insensitive match
  - /all/i matches "Allison Obourn", "small", "JANE GOODALL", ...

Special characters: |, (), \

- | means *OR*
  - /abc|def|g/ matches "abc", "def", or "g"
  - There's no *AND* symbol. Why not?
- () are for grouping
  - /(Homer|Marge) Simpson/ matches "Homer Simpson" or "Marge Simpson"
- \ starts an escape sequence
  - many characters must be escaped to match them literally: / \ $. [ ] ( ) ^ * + ?
  - /<br \/>/ matches lines containing <br /> tags

Quantifiers: *, +, ?

- * means 0 or more occurrences

  - /abc*/ matches "ab", "abc", "abcc", "abccc", ...

  - /a(bc)*/ matches "a", "abc", "abcbc", "abcbcbc", ...

  - /a.*a/ matches "aa", "aba", "a8qa", "a!?xyz__9a", ...

- \+ means 1 or more occurrences

  - /Hi!+ there/ matches "Hi! there", "Hi!!! there", ...

  - /a(bc)+/ matches "abc", "abcbc", "abcbcbc", ...

- ? means 0 or 1 occurrences

  - /a(bc)?/ matches "a" or "abc"

More quantifiers: {min,max}

- {*min*,*max*} means between *min* and *max* occurrences (inclusive)
    - /a(bc){2,4}/ matches "abcbc", "abcbcbc", or "abcbcbcbc"
- *min* or *max* may be omitted to specify any number
    - {2,} means 2 or more
    - {,6} means up to 6
    - {3} means exactly 3

Practice exercise

- When you search Google, it shows the number of pages of results as "o"s in the word "Google". What regex matches strings like "Google", "Gooogle", "Goooogle", ...? (try it) (data)

- Answer: /Goo+gle/ (or /Go{2,}gle/)

Anchors: ^ and $

- ^ represents the beginning of the string or line;
  $ represents the end

  - /Jess/ matches all strings that contain Jess;
    /^Jess/ matches all strings that *start with* Jess;
    /Jess$/ matches all strings that *end with* Jess;
    /^Jess$/ matches the exact string "Jess" only

  - /^Alli.*Obourn$/ matches "AlliObourn", "Allie Obourn", "Allison E Obourn", ...
    but NOT "Allison Obourn stinks" or "I H8 Allison Obourn"

- (on the other slides, when we say, /PATTERN/ matches "text", we really mean that it matches any string that contains that text)

Character sets: []

- [] group characters into a **character set**; will match any single character from the set
  - /[bcd]art/ matches strings containing "bart", "cart", and "dart"
  - equivalent to /(b|c|d)art/ but shorter
- inside [], many of the modifier keys act as normal characters
  - /what[!*?]*/ matches "what", "what!", "what?**!", "what??!", ...
- What regular expression matches DNA (strings of A, C, G, or T)?
  - /[ACGT]+/

Character ranges: [start-end]

- inside a character set, specify a range of characters with -
    - /[a-z]/ matches any lowercase letter
    - /[a-zA-Z0-9]/ matches any lower- or uppercase letter or digit
- an initial ^ inside a character set negates it
    - /[^abcd]/ matches any character other than a, b, c, or d
- inside a character set, - must be escaped to be matched
    - /[+\-]?[0-9]+/ matches an optional + or -, followed by at least one digit

Practice Exercises

What regular expression matches letter grades such as A, B+, or D- ? (try it) (data)

What regular expression would match UW Student ID numbers? (try it) (data)

What regular expression would match a sequence of only consonants, assuming that the string consists only of lowercase letters? (try it) (data)

Escape sequences

- special escape sequence character sets:
  - \d matches any digit (same as [0-9]); \D any non-digit ([^0-9])
  - \w matches any word character (same as [a-zA-Z_0-9]); \W any non-word char
  - \s matches any whitespace character (, \t, \n, etc.); \S any non-whitespace

- What regular expression matches names in a "Last, First M." format with any number of spaces?

  - /\w+,\s+\w+\s+\w\./

Regular expressions in PHP (PDF)

- regex syntax: strings that begin and end with /, such as "/[AEIOU]+/"

function	description
preg_match(*regex, string*)	returns TRUE if *string* matches *regex*
preg_replace(*regex, replacement, string*)	returns a new string with all substrings that match *regex* replaced by *replacement*
preg_split(*regex, string*)	returns an array of strings from given *string* broken apart using given *regex* as delimiter (like explode but more powerful)

PHP form validation w/ regexes

```php
$state = $_POST["state"];
if (!preg_match("/^[A-Z]{2}$/", $state)) {
  print "Error, invalid state submitted.";
}                                         PHP
```

- preg_match and regexes help you to validate parameters

- sites often *don't* want to give a descriptive error message here (why?)

Regular expression PHP example

```php
# replace vowels with stars
$str = "the quick    brown        fox";

$str = preg_replace("/[aeiou]/", "*", $str);
                              # "th* q**ck    br*wn        f*x"

# break apart into words
$words = preg_split("/[ ]+/", $str);
                              # ("th*", "q**ck", "br*wn", "f*x")

# capitalize words that had 2+ consecutive vowels
for ($i = 0; $i < count($words); $i++) {
  if (preg_match("/\\\*{2,}/", $words[$i])) {
    $words[$i] = strtoupper($words[$i]);
  }
}                             # ("th*", "Q**CK", "br*wn", "f*x")      PHP
```

The die function

```php
die("error message text");
```

- PHP's die function prints a message and then completely stops code execution

- it is sometimes useful to have your page "die" on invalid input

- problem: poor user experience (a partial, invalid page is sent back)

The header function

```php
header("HTTP header text");    # in general
header("Location: url");       # for browser redirection        PHP
```

- PHP's header function can be used for several common HTTP messages

  - sending back HTTP error codes (404 not found, 403 forbidden, etc.)

  - redirecting from one page to another

  - indicating content types, languages, caching policies, server info, ...

- you can use a Location header to tell the browser to redirect itself to another page

  - useful to redirect if the user makes a validation error

  - **must** appear before any other HTML output generated by the script

Using header to redirect between pages

```php
header("Location: url");                                    PHP
```

```php
$city  = $_POST["city"];
$state = $_POST["state"];
$zip   = $_POST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) != 5) {
  header("Location: start-page.php");    # invalid input; redirect
}                                                           PHP
```

- *one problem*: User is redirected back to original form without any clear error message or understanding of why the redirect occurred. (We can improve this later.)

Handling invalid data

```php
function check_valid($regex, $param) {
  if (preg_match($regex, $_POST[$param])) {
    return $_POST[$param];
  } else {
    # code to run if the parameter is invalid
    die("Bad $param");
  }
}
...
$sid     = check_valid("/^[0-9]{7}$/", "studentid");
$section = check_valid("/^[AB][A-C]$/i", "section");
```
PHP

- Having a common helper function to check parameters is useful.
- If your page needs to show a particular HTML output on errors, the die function may not be appropriate.

Regular expressions in HTML forms

```
How old are you?

<input type="text" name="age" size="2" pattern="[0-9]+" title="an integer" />

<input type="submit" />                                        HTML
```

How old are you? [   ] [Submit Query]   output

- HTML5 adds a new pattern attribute to input elements

- the browser will refuse to submit the form unless the value matches the regex