

# Web Programming Step by Step

## Lecture 11 Form Validation

Except where otherwise noted, the contents of this presentation are Copyright 2009 Marty Stepp and Jessica Miller.



---

## 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



Secure Site

Questions? Call us:

(800) 788-7000

## Signing Up is Easy

 Some of the information you entered is missing or incorrect. Please check all highlighted fields.

-  Please enter Last Name using letters, apostrophes or dashes.
-  Enter a valid date for Date of Birth.
-  Please enter a valid e-mail address.

### Personal Info

First Name:

Last Name:

Date of Birth:

E-mail Address:

Identify yourself by your:

- Account Number
- ATM/Debit Card
- Credit Card

# 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

---

```
<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>
```

HTML

```
City: 
State: 
ZIP: 

```

output

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

---

## Basic server-side validation code

---

```
$city = $_REQUEST["city"];
$state = $_REQUEST["state"];
$zip = $_REQUEST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) != 5) {
  ?>
  <h2>Error, invalid city/state submitted.</h2>
  <?php
}
```

PHP

- *basic idea*: examine parameter values, and if they are bad, show an error message and abort
- validation code can take a lot of time / lines to write
  - 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?)

---

# What is a regular expression?

---

```
"/^[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

---

## 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
  - `"/mart/i"` matches "Marty Stepp", "smart fellow", "WALMART", ...

---

## 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"
- `^` matches the beginning of a line; `$` the end
  - `"/^<!--$/"` matches a line that consists entirely of "`<!--`"
- `\` 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!?!\_a", ...
- + means 1 or more occurrences
  - `"/a(bc)+/` matches "abc", "abcbc", "abcbcbc", ...
  - `"/Goo+gle/` matches "Google", "Gooogle", "Gooooogle", ...
- ? 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

---

## 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
- What regular expression matches letter grades such as A, B+, or D-?
  - `"/[ABCDF][+\-]?/"`

---

## 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 dollar amounts of at least \$100.00 ?
  - `"/\$\d{3,}\.\d{2}/"`

---

## Regular expressions in PHP (PDF)

---

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

function	description
<code>preg_match(<i>regex</i>, <i>string</i>)</code>	returns TRUE if <i>string</i> matches <i>regex</i>
<code>preg_replace(<i>regex</i>, <i>replacement</i>, <i>string</i>)</code>	returns a new string with all substrings that match <i>regex</i> replaced by <i>replacement</i>
<code>preg_split(<i>regex</i>, <i>string</i>)</code>	returns an array of strings from given <i>string</i> broken apart using the given <i>regex</i> as the delimiter (similar to <code>explode</code> but more powerful)

---

## Regular expression example

---

```
# 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

- notice how \ must be escaped to \\

---

## PHP form validation w/ regexes

---

```
$state = $_REQUEST["state"];
if (!preg_match("/[A-Z]{2}/", $state)) {
?>

    <h2>Error, invalid state submitted.</h2>

<?php
}
```

PHP

- using preg\_match and well-chosen regexes allows you to quickly validate query parameters against complex patterns