

CSCI 241

Validating User Input

Department of CS & QM
Winthrop University
Fall 2011

Prepared by X. Wang

Topics

- Regular Expressions
- Server-Side Validation in PHP
- Client-Side Validation in JavaScript

Prepared by X. Wang

2

Regular Expressions

- Regular expressions provide the foundation for pattern-matching functionality
- Regular expressions can achieve more sophisticated pattern matching to find, extract and replace complex substrings within a string
- The functions with regular expressions usually are not as efficient as the simple functions described in strings

Prepared by X. Wang

3

Regular Expressions

- PHP offers two sets of regular expression functions: POSIX (Portable Operating System Interface for uniX) and PCRE (Perl Compatible Regular Expression)
- Both sets are very similar
- Here we will have a brief description of the POSIX regular expression syntax
- More information can be found on the Internet
- The POSIX regular expression related functions are listed at <http://www.php.net/manual/en/ref.regex.php>

Prepared by X. Wang

4

Regular Expressions

- To demonstrate the syntax of regular expressions, we will use the function `ereg()`:
boolean ereg(string pattern, string source [, array var])
 - `ereg()` returns `true` if the regular expression `pattern` is found in the `source` string
 - The optional argument will be discussed later on
 - The following trivial example shows how `ereg()` works
- ```
if (ereg("cat", "raining cats and dogs"))
 print "Found \"cat\"";
```

Prepared by X. Wang

5

## Regular Expressions

---

- ☐ Characters and wildcards
  - To represent any character in a pattern, a period (`.`) is used as a wildcard
  - To express a pattern that actually matches a period character, we need to use the backslash character
- ```
$found = ereg("c.", "raining cats and dogs");
// $found is set to true
$found = ereg("\.edu", "www.winthrop.edu");
// $found is set to true
```

Prepared by X. Wang

6

Regular Expressions

- Character lists
- A list of characters enclosed in **brackets** can be specified within a pattern
- A list can specify characters that are not matches using the **not operator (^)** as the first character in the brackets
- The pattern "**[ABC][0-9]**" specifies a pattern with only **two** characters. The first one is A, B or C; the second one is a **digit**
- The pattern "**p[^h]p**" specifies a pattern with three characters. The first and third characters are p, and the second character is **any character other than h**

Prepared by X. Wang

7

Regular Expressions

- Anchors
- A regular expression can specify that a pattern occurs at the start or end of a source string using **anchors**
- The **^** character anchors a pattern to the start (don't confuse this with the **not** in the previous character list)
- The **\$** character anchors a pattern to the end of a string

```
$var = "to be or not to be";
$match = ereg('^to', $var); // true
$match = ereg('be$', $var); // true
$match = ereg('or', $var); // false

$var = "1234567";
$match = ereg('[0-9]', $var); // true
$match = ereg('[^0-9]', $var); // false
```

Prepared by X. Wang

8

Regular Expressions

- Repeating characters
- A **?** specifies to match the preceding character **zero or one** times
- A **+** specifies to match the preceding character **one or more** times
- A ***** specifies to match the preceding character **zero or more** times
- A **curly braces with a number** specifies the **fixed number of occurrences**

Prepared by X. Wang

9

Regular Expressions

- Groups
- Sub-patterns in a regular expression can be grouped with **parentheses** around them
- Example:

```
$pattern = '^(\http://)?[a-zA-Z]+(\.[a-zA-Z]+)$';
$found = ereg($pattern, "www.winthrop.edu");
// true
```

Prepared by X. Wang

10

Regular Expressions

- Alternative patterns
- **Alternatives** in a pattern are specified with the **| operator**
- Example:

```
$pattern = '(com$|edu$|net$|org$)';
$found = ereg($pattern, "www.winthrop.edu"); // true
$found = ereg($pattern, "www.oracle.com"); // true
$found = ereg($pattern, "www.oracle.com.cn"); // false
$found = ereg($pattern, "www.php.net"); // true
$found = ereg($pattern, "www.abcd.gov"); // false
```

Prepared by X. Wang

11

Regular Expressions

- Escaping special characters
- Escaping the special meaning of a character is done with the **backslash**
- An example: the pattern "**2\+3**" matches the string "**2+3**"
- If the **+** isn't escaped, the pattern matches one or many occurrences of the character **2** followed by **3**
- Another way is to place the **+** in the list as "**2[+]**"

Prepared by X. Wang

12

Regular Expressions

- Metacharacters
- Metacharacters such as `\t`, `\d`, `\s`, `\n` etc. can be used in regular expressions


```
$source = "fast\tfood";
$result = ereg('\s', $source); // true
```
- Special metacharacters in the form `[:...:]` can also be used in character lists to match other characters


```
$str = "abcd1234";
$result = ereg('^[[:alnum:]]+$', $str); // true
```

Prepared by X. Wang 13

Regular Expressions

- Metacharacters
- Pattern Matches

| | |
|-------------------------|---|
| <code>[:alnum:]</code> | Letters and digits |
| <code>[:alpha:]</code> | Letters |
| <code>[:blank:]</code> | The space and Tab characters |
| <code>[:cntrl:]</code> | Control characters |
| <code>[:digit:]</code> | Digits. Equivalent to <code>\d</code> |
| <code>[:graph:]</code> | Characters represented with a visible character |
| <code>[:lower:]</code> | Lowercase letters |
| <code>[:print:]</code> | Include <code>[:graph:]</code> and <code>[:blank:]</code> |
| <code>[:space:]</code> | Whitespace character. Equivalent to <code>\s</code> |
| <code>[:upper:]</code> | Uppercase letters |
| <code>[:xdigit:]</code> | Hexadecimal digits |

Prepared by X. Wang 14

Regular Expression Functions

- Finding and extracting values
- The `ereg()` function, and the case-insensitive version `eregi()`, are defined as


```
boolean ereg(string pattern, string source [, array var])
boolean eregi(string pattern, string source [, array var])
```
- Both functions return `true` if the regular expression `pattern` is found in the `source` string
- Both functions return `false` if the `pattern` is not found
- The optional argument `var` is populated with the portions of `source` that are matched by up to nine grouped subexpressions in `pattern`

Prepared by X. Wang 15

Regular Expression Functions

- Finding and extracting values
- Sub expressions consist of characters enclosed in parentheses
- An example:


```
$pattern = '^[0-9]{4}-([0-9]{2})-([0-9]{2})$';
$value = "2011-10-01";
ereg($pattern, $value, $var);
print_r($var);
print "<br />";
```

The output is

```
Array ( [0] => 2011-10-01 [1] => 2011 [2] => 10 [3] => 01 )
```

Prepared by X. Wang 16

Regular Expression Functions

- Replacing substrings
- The following functions create new strings by replacing substrings


```
string ereg_replace(string pattern, string replace, string source)
string eregi_replace(string pattern, string replace, string source)
```
- They create a new string by replacing substrings of the `source` string that match the regular expression `pattern` with a `replace` string
- ```
$pattern = '^[0-9]{4}-([0-9]{2})-([0-9]{2})$';
$value = "2011-10-01";
print ereg_replace($pattern, '\3/\2/\1', $value);
```

The output is: 01/10/2011

Prepared by X. Wang 17

## Regular Expression Functions

- Splitting a string into an array
- The following functions split strings
 

```
array split(string pattern, string source [, integer limit])
array spliti(string pattern, string source [, integer limit])
```
- They split the `source` string into an array, breaking the string where the matching `pattern` is found
- The `limit` determines the maximum number of elements in the returned array
- They perform a similar task to the `explode()`

Prepared by X. Wang 18

## Regular Expression Functions

- Splitting a string into an array

- An example

```
$sentence = "I wonder why he does\nBuzz, buzz, buzz";
$words = split("[^a-zA-Z]+", $sentence);
print_r($words);
```

- The output is:

```
Array ([0] => I [1] => wonder [2] => why [3] => he [4]
=> does [5] => Buzz [6] => buzz [7] => buzz)
```

Prepared by X. Wang

19

## Validation and Error Reporting

- User data validation is essential to web database application
- We have already seen how to check empty fields of a form in some examples
- Now let's introduce the principles of data validation
- Then we introduce how to validate strings, numbers, dates, times, and email addresses in PHP at the server-side
- Finally, we mention the validation inJavaScript at the client-side

Prepared by X. Wang

20

## Validation and Error Reporting Principles

- Validation has actually two processes: **finding errors** and **presenting error messages**
- **Finding errors** can be **interactive**, where data is checked as it is entered, or **post-validation**, where the data is checked after entry
- **Presenting errors** can be **field-by-field**, where a new error message is presented to the user for each error found, or **batched**, where all errors are presented as a single message

Prepared by X. Wang

21

## Validation and Error Reporting Principles

- For considering only the basic processes, the choice of when to check errors and when to notify the user leads to four common approaches (models):
  - Interactive validation with field-by-field errors
  - Interactive validation with batched errors
  - Post-validation with field-by-field errors
  - Post-validation with batched errors

Prepared by X. Wang

22

## Validation and Error Reporting Principles

- Interactive models are difficult to be implemented by the scripts at the server-side in the web environment
- To do this, an HTTP request and response is required to validate each field that is entered
- This is usually unacceptable because the user is required to submit the data after entering all fields
- Client-side scripts can implement an interactive model
- However, validation at the client-side should not be the only method of validation because the user can **passively or actively bypass** the client-side processes

Prepared by X. Wang

23

## Validation and Error Reporting Principles

- Post-validation models are practical in web database applications
- Both client- and server-side scripts can validate all form data during the submission process
- In many applications, reasonably comprehensive validation is performed at the client-side when the user clicks the form submit button
- Client-side validation reduces server and network load
- Client-side validation is also usually faster for the user

Prepared by X. Wang

24

## Validation and Error Reporting Principles

- If client-side validation succeeds, data is submitted to the server and the same (or often more comprehensive) validation is performed
- Duplicating client validation on the server is essential because of the unreliability of client-side scripts and lack of control over the client environment
- Client-side validation may be necessary since that is faster, but never trust the user's data or the client browser

Prepared by X. Wang

25

## Validation and Error Reporting Principles

- The post-validation model can be combined with either field-by-field or batched error reporting
- For server-side validation, the batched model is preferable to a field-by-field implementation
- For client-side post-validation, either field-by-field or batched model can be used

Prepared by X. Wang

26

## Server-Side Validation in PHP

- Now let's see how to implement server-side validation in PHP
- We will discuss how to use PHP to check mandatory fields, field lengths and data types, and to validate numbers including currencies and credit cards, strings including email addresses and Zip codes, and dates and times
- The PHP functions that will be used include the regular expression and string processing functions

Prepared by X. Wang

27

## Server-Side Validation in PHP

- Mandatory Data
- Testing whether mandatory fields have been entered is straightforward
- We have used this testing in implementing the register operation for our online bookstore
- For example:
 

```
if (!isset($lastname)||$lastname=="")
 die("The Last Name field cannot be blank");
or
if (!isset($lastname)||empty($lastname))
 print "The Last Name field cannot be blank";
```

Prepared by X. Wang

28

## Server-Side Validation in PHP

- Validating Strings
- Strings are basic data type in PHP
- For example, the user data stored in the `$_GET` or `$_POST` array are strings
- A lot of data retrieved from a database table are also strings
- The simplest test of a string is to check whether it meets a minimum or maximum length requirements
- For example,
 

```
if (strlen($password)<4 || strlen($password)>8)
 print "Password must contain between 4 and 8 characters";
```

Prepared by X. Wang

29

## Server-Side Validation in PHP

- Validating Strings
- Other common tests for strings include checking if strings are uppercase, lowercase, alphabetic or drawn from a defined character set
- To test if a string is alphabetic:
 

```
if (ereg("[[:alpha:]]+$", $string)) // if (ereg("[a-z]+$", $string))
 print "String must contain only alphabetic characters";
```
- To test if a string is uppercase:
 

```
if (ereg("[[:upper:]]+$", $string))
 print "String can contain only uppercase characters";
```
- To test if a string is lowercase:
 

```
if (ereg("[[:lower:]]+$", $string))
 print "String can contain only lowercase characters";
```

Prepared by X. Wang

30

## Server-Side Validation in PHP

### Validating Strings

- To test if a student ID is beginning with an S:
 

```
if (!ereg("S", $studentID))
 print "Student ID must begin with an S";
```

or

```
if (substr($studentID, 0, 1) != "S")
 print "Student ID must begin with an S";
```
- To test if a first name contains only letters, apostrophes, spaces and hyphens :
 

```
if (!ereg("{a-z' -}+", $firstname))
 print "First name can contain only letters or ' or spaces or -";
```

Prepared by X. Wang

31

## Server-Side Validation in PHP

### Validating Zip codes

- To test if a zipcode is five digits
 

```
if (!ereg("[0-9]{5}", $zipcode))
 print "Zipcode must contain only five digits";
```

or

```
if (!(ereg("[0-9]{5}", $zipcode) || ereg("[0-9]{5}-[0-9]{4}",
 $zipcode)))
 print "Zipcode can contain only 5 digits, or - with 4 more digits";
```

Prepared by X. Wang

32

## Server-Side Validation in PHP

### Validating email address

- Validating an email address is complex
- We usually need to do the following three validations
  - Check whether the address is empty
  - Check whether the address violates the format
  - Check whether the address exceeds the field length
- Or if possible, we can also check whether the domain name is correct

Prepared by X. Wang

33

## Server-Side Validation in PHP

### Validating email address

- The following is a function to check an email address

```
function checkEmail($email) {
 $validEmailExpr = "[0-9a-z-!#$%&_]+(\.[0-9a-z-!#$%&_]+)*" .
 "@[0-9a-z-!#$%&_]+(\.[0-9a-z-!#$%&_]+)+$";
 if (empty($email)) {
 print "The email field cannot be blank";
 return false;
 }
 elseif (!ereg($validEmailExpr, $email)) {
 print "The email address must be in the name@domain format";
 return false;
 }
 elseif (strlen($email) > 50) {
 print "The email address can be no longer than 30 characters";
 return false;
 }
 return true;
}
```

Prepared by X. Wang

34

## Server-Side Validation in PHP

### Validating URLs

- To validate a URL, we can use the `parse_url()` function
- This function returns an associative array including elements with the keys: `scheme`, `host`, `path`, `query`, etc.
- For example, to validate a URL in `$url`, we can do
 

```
$urlArray = parse_url($url);
if ($urlArray["scheme"] != "http")
 print "URL must begin with http://";
elseif (empty($urlArray["host"]))
 print "URL must include a host name";
```

Prepared by X. Wang

35

## Server-Side Validation in PHP

### Validating Numbers

- To check if a string contains only digits (including point), we can use the `is_numeric()` function
- To convert a digital string into value, we can use the functions `intval()` or `floatval()`
- For example, to get and validate a salary, we can do
 

```
if (!is_numeric($_POST["salary"]))
 print "Salary must be numeric
";
else
 $salary = floatval($_POST["salary"]);
```

Prepared by X. Wang

36

## Server-Side Validation in PHP

### Validating Currencies

- To validate a currency, we can use regular expression
- For example, to get and validate a price, we can do

```
$priceStr = $_POST["price"];
if (!is_numeric($priceStr))
 print "Price must be numeric
";
elseif (!ereg("[0-9]{1,3}[.][0-9]{2}$", $priceStr))
 print "Price must be between $0.00 and $999.00
";
else
 $price = floatval($priceStr);
```

Prepared by X. Wang

37

## Server-Side Validation in PHP

### Validating Phone numbers

- We can also use regular expression to do the validation
- For example,

```
$validPhone = "^[0-9]{3}([0-9]{3})?[-]?[0-9]{3}[-]?[0-9]{4}$";
if (empty($phone))
 print "The phone number cannot be blank";
elseif (!ereg($validPhone, $phone))
 print "The phone number must be 7 digits with an optional 3
 digit area code
";
```

Prepared by X. Wang

38

## Server-Side Validation in PHP

### Validating Credit Card numbers

- A credit card number validation is very important for an online shopping store, and usually includes three steps
  - Check whether the card number contains only digits and spaces, and then removes the spaces
  - Extract the first one, two, three or four digits to determine the credit card type, and the correct length of the number. Reject the number that doesn't match any credit card type, and the length of the required number
  - Use the Luhn algorithm to validate the number

Prepared by X. Wang

39

## Server-Side Validation in PHP

### Validating Credit Card numbers

- The first step is straightforward
- We can use regular expression to check if the input number contains only digits and spaces, and remove the spaces with str\_replace() function

Prepared by X. Wang

40

## Server-Side Validation in PHP

### Validating Credit Card numbers

- For the second step, we need to know the prefix number for different credit cards
- The following list shows the prefix and length of some credit cards

| Card Type        | Prefix | Length |
|------------------|--------|--------|
| MasterCard       | 51-55  | 16     |
| VISA             | 4      | 13, 16 |
| American Express | 34, 37 | 15     |
| Discover         | 6011   | 16     |

Prepared by X. Wang

41

## Server-Side Validation in PHP

### Validating Credit Card numbers

- The last step, we need to use the Luhn algorithm to verify whether the number is correct
- The Luhn algorithm works as follows
  - Sum up every second digit in the card number, beginning with the last digit and proceeding right-to-left
  - Sum up the double of every second digit in the card number, beginning with the second digit on the right and proceeding right-to-left. If the double of the digit is larger than 9, subtract 9 from the double value before adding it to the sum
  - Determine if the sum of the two steps is a multiple of 10. If it is, the card number is valid, otherwise the number is rejected

Prepared by X. Wang

42

## Server-Side Validation in PHP

### □ Validating Credit Card numbers

- The following is the function to validate a credit card

```
function checkCreditCard($cardnum) {
 // the first step
 if (empty($cardnum)) return 1; // blank field
 if (ereg("[0-9]*$", $cardnum)) return 2; // not contain only digits and spaces
 $cardnum = str_replace(" ", "", $cardnum);
 // the second step
 $stype = "";
 $length = 0;
 $firstOne = intval(substr($cardnum, 0, 1));
 $firstTwo = intval(substr($cardnum, 0, 2));
 $firstFour = intval(substr($cardnum, 0, 4));
 if ($firstTwo >= 51 && $firstTwo <= 55) {
 $stype = "MasterCard"; $length=16;
 }
}
```

Prepared by X. Wang

43

## Server-Side Validation in PHP

### □ Validating Credit Card numbers

```
elseif ($firstOne == 4) {
 $stype = "VISA"; $length=16; // or 13
}
elseif ($firstTwo==34 || $firstTwo==37) {
 $stype = "American Express"; $length=15;
}
elseif ($firstFour==6011) {
 $stype = "Discover"; $length=16;
}

if (empty($stype)) return 3; // invalid card number
if (strcmp($stype, "VISA") == 0) {
 $length = strlen($cardnum);
 if ($length==16 && $length!=13) return 3;
}
elseif ($length != strlen($cardnum)) return 3;
```

Prepared by X. Wang

44

## Server-Side Validation in PHP

### □ Validating Credit Card numbers

```
// The third step
$check = 0;
for ($i=$length-1; $i>=0; $i-=2)
 $check += intval(substr($cardnum, $i, 1));
for ($i=$length-2; $i>=0; $i-=2) {
 $double = intval(substr($cardnum, $i, 1)) * 2;
 $check += $double;
 if ($double >= 10) $check -= 9;
}
if ($check % 10 != 0) return 3;
return 0; // card number is valid
}
```

Prepared by X. Wang

45

## Server-Side Validation in PHP

### □ Validating Dates

- Because the input of a date can be different formats, validating dates also becomes complicated
- Usually, we specify one input format for the user to follow
- For example, `mm/dd/yyyy` is a normal format
- We usually use [regular expression](#) to validate the input format, and then call the `checkdate()` function to verify whether the date numbers (month, day, and year) are valid

Prepared by X. Wang

46

## Server-Side Validation in PHP

### □ Validating Dates

- The following is a function to validate a date

```
function checkmydate($adate) {
 if (empty($adate)) {
 print "The Date field cannot be blank"; return false;
 }
 elseif (ereg("^[0-9]{2}/[0-9]{2}/[0-9]{4}$", $adate, $parts)) {
 print "The entered date is not valid in the format MM/DD/YYYY";
 return false;
 }
 elseif (!checkdate($parts[1], $parts[2], $parts[3])) {
 print "The date is invalid. Please check the month and day";
 return false;
 }
 return true;
}
```

Prepared by X. Wang

47

## Server-Side Validation in PHP

### □ Validating Times

- Time validation is similar to the Data validation
- We can still use regular expression to validate the time input format
- Then we can check whether the time numbers (hour, minute, and second) are valid. For this step, we need to write code to verify the time numbers. There is no existing function to call for this validation in PHP

Prepared by X. Wang

48

### Client-Side Validation in JavaScript

- JavaScript is a script language that can be used to validate the data entries at the client-side
- Client-side validation in JavaScript is **optional**, but has benefits
- It can provide **faster response** to the user than server-side validation, and **reduce network traffic and server load**
- Also, it can be implemented as **interactive validation** (where errors are checked as they occur) and **field-by-field reporting** (where error messages are shown individually)

Prepared by X. Wang

49

### Client-Side Validation in JavaScript

- However, validation at the client-side is **unreliable**: the user can bypass the validation through design, error, or misconfiguration of their web browser
- For this reason, **client-side validation** should be used only to increase speed, reduce load, and add features, and **never to replace server-side validation**

Prepared by X. Wang

50

### Client-Side Validation in JavaScript

- How to implement client-side validation in JavaScript should have been discussed in INF141
- In my online bookstore demo website at <http://infd.birdnest.org/bookstore/register.php>, you can find how I used JavaScript to validate the user's input for the Register operation

Prepared by X. Wang

51

### Client-Side Validation in JavaScript

- The following is the HTML form used in the `register.php` page for the Register operation

```
<table width="100%" border="0" cellpadding="0" cellspacing="0">
<form onSubmit="return(validate());" name="form1" id="form1"
method="post" action="saveregister.php">
<tr>
<td width="100" height="36"> </td>
<td width="100"> </td>
<td width="450"> </td>
</tr>
<tr>
<td height="40"> </td>
<td colspan="2" valign="top">To obtain an account, please
provide the following information

(* is a required field)</td>
</tr>
```

Prepared by X. Wang

52

### Client-Side Validation in JavaScript

```
<tr><td colspan="3" height="18"> </td></tr>
<tr>
<td colspan="2" height="20" align="right">Last Name(*):</td>
<td><input type="text" name="lastname" size="30" /></td></tr>
<tr>
<td colspan="2" height="20" align="right">First Name(*):</td>
<td><input type="text" name="firstname" size="30" /></td></tr>
<tr>
<td colspan="2" height="20" align="right">Email(*):</td>
<td><input type="text" name="email" size="30" /></td></tr>
<tr>
<td colspan="2" height="20" align="right">Password(*):</td>
<td><input type="password" name="password1" size="30" /></td>
</tr>
<tr>
<td colspan="2" height="20" align="right">Confirm Password(*):
</td>
<td><input type="password" name="password2" size="30" /></td>
</tr>
```

Prepared by X. Wang

53

### Client-Side Validation in JavaScript

```
<tr>
<td colspan="2" height="20" align="right">Address(*):</td>
<td><input type="text" name="address" size="60" /></td>
</tr>
<tr>
<td colspan="2" height="20" align="right">City(*):</td>
<td><input type="text" name="city" size="17" /></td>
</tr>
<tr>
<td colspan="2" height="20" align="right">State(*):</td>
<td><select name="state">
<?php
require "db.php";
foreach ($states as $state) {
print "<option value=\"\".$state.\"\">.$state."</option>";
}
?></select></td>
</tr>
```

Prepared by X. Wang

54

### Client-Side Validation in JavaScript

```

<tr>
 <td colspan="2" height="20" align="right">Zipcode(*):</td>
 <td><input type="text" name="zipcode" size="17" /></td>
</tr>
<tr>
 <td colspan="2" height="20" align="right">Phone:</td>
 <td><input type="text" name="phone" size="17" /></td>
</tr>
<tr>
 <td colspan="3" height="18"> </td>
</tr>
<tr>
 <td colspan="2" height="20"> </td>
 <td><input type="submit" value="Submit" /> </td>
 </td> <input type="reset" value="Cancel" /></td>
</tr>
<tr>
 <td colspan="3" height="202"> </td>
</tr>
</form></table>

```

Prepared by X. Wang

55

### Client-Side Validation in JavaScript

- The following is the JavaScript function in the `<head></head>` block to validate the user's input items

```

<script type="text/javascript">
<!--
function validate() {
 if (form1.lastname.value.length == 0) {
 alert("The Last Name field cannot be blank");
 return false;
 }
 if (form1.firstname.value.length == 0) {
 alert("The First Name field cannot be blank");
 return false;
 }
}

```

Prepared by X. Wang

56

### Client-Side Validation in JavaScript

```

if (form1.email.value.length == 0) {
 alert("The Email field cannot be blank");
 return false;
}
if (form1.password1.value.length == 0) {
 alert("The Password field cannot be blank");
 return false;
}
if (form1.password2.value.length == 0) {
 alert("The Confirm Password field cannot be blank");
 return false;
}
if (form1.address.value.length == 0) {
 alert("The Address field cannot be blank");
 return false;
}
if (form1.city.value.length == 0) {
 alert("The City field cannot be blank");
 return false;
}
}

```

Prepared by X. Wang

57

### Client-Side Validation in JavaScript

```

if (form1.state.value.length == 0) {
 alert("The State field cannot be blank");
 return false;
}
if (form1.zipcode.value.length == 0) {
 alert("The Zipcode field cannot be blank");
 return false;
}
return true;
}
// -->
</script>

```

Prepared by X. Wang

58