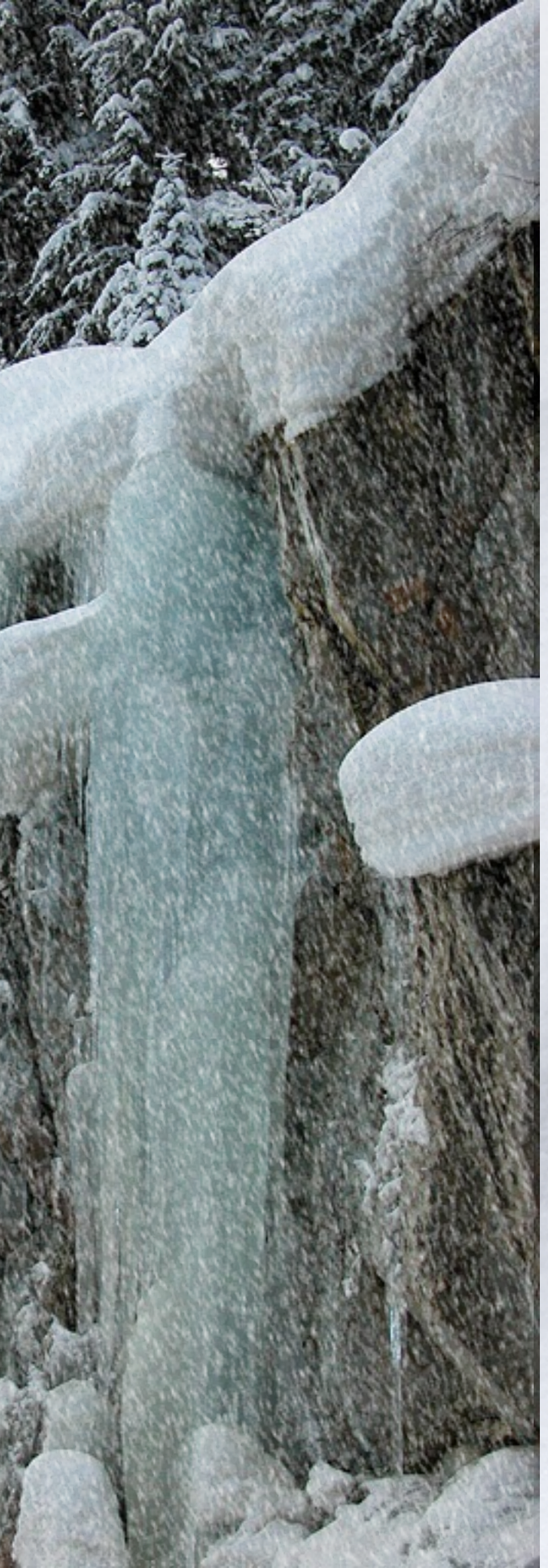
Lecture 8
More PHP for Server Side Programming

Play



Outline

- Submitting Data
- Processing Form data in PHP
- Form Validations
- Regular Expression
- Object-Oriented PHP

Example: Form

```
<html>
<body>

<form action="welcome.php" method="post">
    Name: <input type="text" name="name"><br>
    E-mail: <input type="text" name="email"><br>
    <input type="submit">
</form>

</body>
</html>
```

```
<html>
<body>

    Welcome <?php echo $_POST["name"]; ?><br>
    Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

HTTP GET vs. POST requests

- GET: asks a server for a page or data if the request has parameters, they are sent in the URL as a query string
  - Information is visible to everyone.
  - GET also has limits on the amount of information to send. The limitation is about 2000 characters.
  - It is possible to bookmark the page.
  - GET may be used for sending non-sensitive data. GET should NEVER be used for sending passwords or other sensitive information!
- POST: submits data to a web server and retrieves the server's response if the request has parameters, they are embedded in the request's HTTP packet, not the URL
  - Information sent is invisible to others (all names/values are embedded within the body of the HTTP request)
  - POST has no limits on the amount of information to send.
  - Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
  - However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
  - Developers prefer POST for sending form data.

URL-encoding

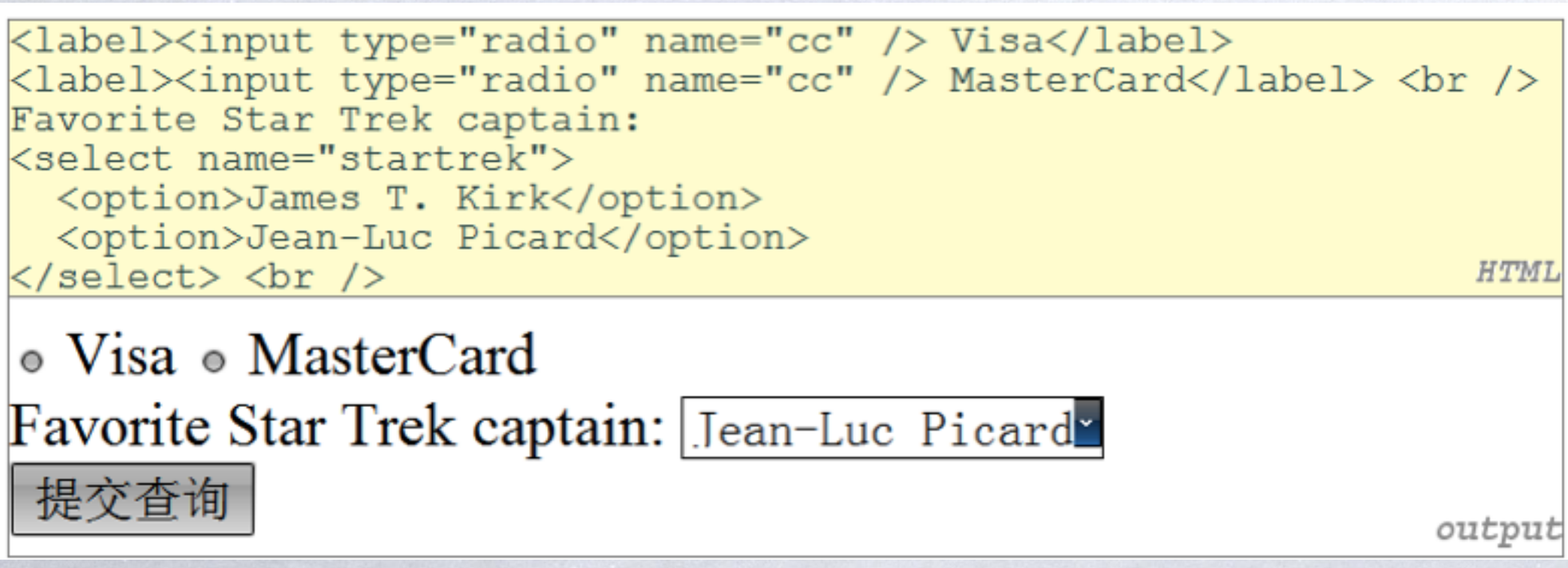
- certain characters are not allowed in URL query parameters:
  - examples: " ", "/", "=", "&"
- when passing a parameter, it is URL-encoded(reference table)
  - "Marty's cool!?" → "Marty%27s+cool%3F%21"
- you don't usually need to worry about this:
  - the browser automatically encodes parameters before sending them
  - the PHP $_REQUEST array automatically decodes them
  - ... but occasionally the encoded version does pop up (e.g. in Firebug)

Problems with submitting data

- this form submits to our handy params.php tester page
- the form may look correct, but when you submit it...

```
<label><input type="radio" name="cc" /> Visa</label>
<label><input type="radio" name="cc" /> MasterCard</label> <br />
Favorite Star Trek captain:
<select name="startrek">
  <option>James T. Kirk</option>
  <option>Jean-Luc Picard</option>
</select> <br />
```

HTML



output

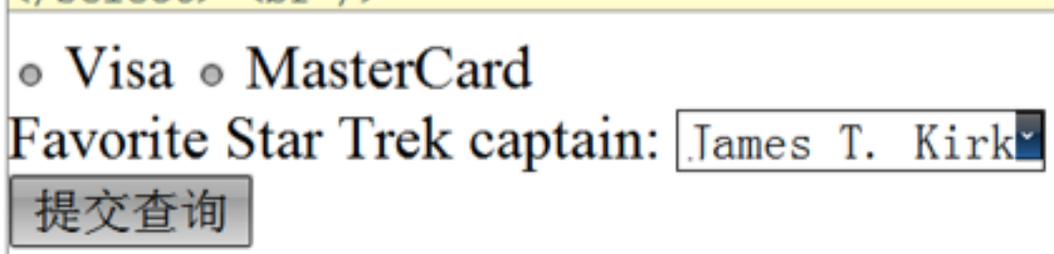
[cc] => on, [startrek] => Jean-Luc Picard

The value attribute

- value attribute sets what will be submitted if a control is selected
- [cc] => visa, [startrek] => kirk

```
<label><input type="radio" name="cc" value="visa" /> Visa</label>
<label><input type="radio" name="cc" value="mastercard" /> MasterCard</label> <br />
Favorite Star Trek captain:
<select name="startrek">
  <option value="kirk">James T. Kirk</option>
  <option value="picard">Jean-Luc Picard</option>
</select> <br />
```

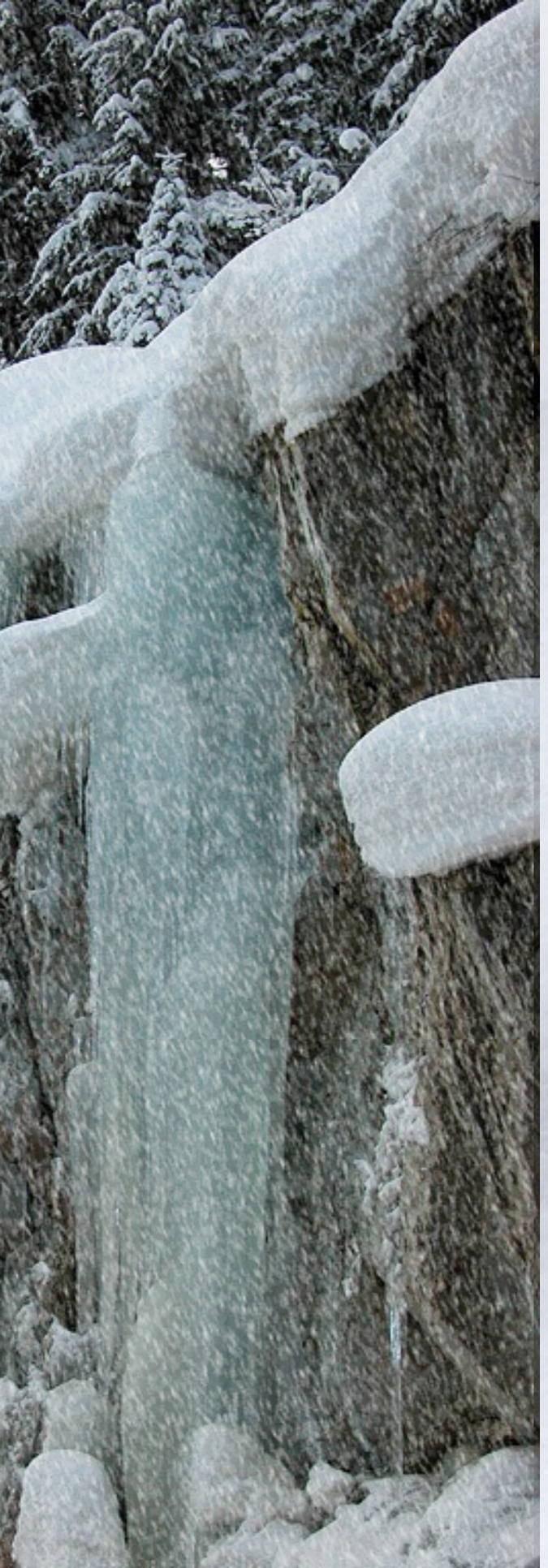
HTML



output

Submitting data to a web server

- though browsers mostly retrieve data, sometimes you want to submit data to a server
  - Hotmail: Send a message
  - Flickr: Upload a photo
  - Google Calendar: Create an appointment
- the data is sent in HTTP requests to the server
  - with HTML forms
  - with predefined URLs
  - with Ajax(seen later)
- the data is placed into the request as parameters



Outline

- Submitting Data
- Processing Form data in PHP
- Form Validations
- Regular Expression
- Object-Oriented PHP

"Superglobal" arrays

- Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- PHP superglobal arrays (global variables) contain information about the current request, server, etc.
- These are special kinds of arrays called associative arrays.

Array	Description
$GLOBALS	to access global variables from anywhere
$_GET, $_POST	parameters passed to GET and POST requests
$_REQUEST	parameters passed to any type of request
$_SERVER, $_ENV	information about the web server
$_FILES	files uploaded with the web request
$_SESSION, $_COOKIE	"cookies" used to identify the user (seen later)

PHP $GLOBALS

- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
- PHP stores all global variables in an array called $GLOBALS[index]. The index holds the name of the variable.

```
<?php
$x = 75;
$y = 25;

function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

PHP $_SERVER

- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

/php/demo_global_server.php
www.w3schools.com
www.w3schools.com
http://www.w3schools.com/php/showphp.asp?filename=demo_global_server
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11) AppleWebKit/601.1.56 (KHTML, like Gecko) Version/9.0 Safari/601.1.56
/php/demo_global_server.php

Processing an uploaded file in PHP

- uploaded files are placed into global array $_FILES, NOT $_REQUEST
- each element of $_FILES is itself an associative array, containing:
  - name: the local filename that the user uploaded
  - type: the MIME type of data that was uploaded, such as image/jpeg
  - size: file's size in bytes
  - tmp_name: a filename where PHP has temporarily saved the uploaded file
    - to permanently store the file, move it from this location into some other file

Uploading details

- 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"

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

浏览… 提交查询

output

Processing uploaded file, example

- 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

```php
$username = $_REQUEST["username"];
if (is_uploaded_file($_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";
}
```

Methods of Processing Form data in PHP

- php.ini, register_globals=On(不建议,会有诸多不安全可能性)
  - variables_order，EGPCS（Environment，GET，POST，Cookie，Server）
- $HTTP_GET_VARS、$HTTP_POST_VARS (不建议)
  - $HTTP_POST_VARS["username"]
- $_POST、$_GET
  - $_POST["username"]
- import_request_variables()
  - import_request_variables("gp", “rvar_")
  - import_request_variables("gp", "")

A Simple HTML Form

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

Welcome John
Your email address is john.doe@example.com

- The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
- The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

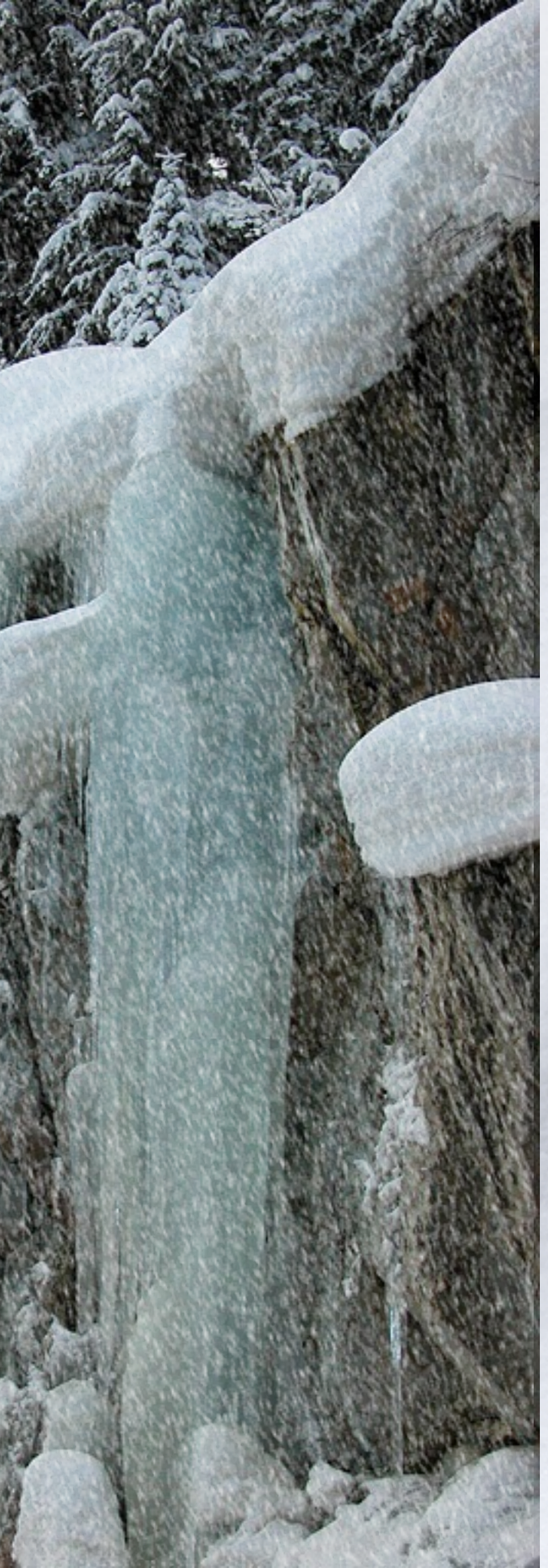
```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

Big Note on PHP Form Security

- The $_SERVER["PHP_SELF"] variable can be used by hackers!
- If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.
- Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.
- Assume we have the following form in a page named "test_form.php":
  - <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
- Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:
  - <form method="post" action="test_form.php">
- However, consider that a user enters the following URL in the address bar:
  - http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
- In this case, the above code will be translated to:
  - <form method="post" action="test_form.php/"><script>alert('hacked')</script>
- This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.
- Be aware of that any JavaScript code can be added inside the <script> tag! A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

- $_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.
- The form code should look like this:
  - <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
- The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:
  - <form method="post" action="test_form.php/ "><script>alert('hacked')</script>">
- The exploit attempt fails, and no harm is done!



Outline

- Submitting Data
- Processing Form data in PHP
- Form Validations
- Regular Expression
- Object-Oriented PHP

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 6-digit integer)
  - ensuring that values fit together (user types email twice, and the two must match)

Client vs. serve-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

PHP Form Validation Example

PHP Form Validation Example

Name:

E-mail:

Website:

Comment:

Gender: Female Male

Submit

Your Input:

```
<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <br><br>
  E-mail: <input type="text" name="email">
  <br><br>
  Website: <input type="text" name="website">
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="fem
  <input type="radio" name="gender" value="mal
  <br><br>
  <input type="submit" name="submit" value="Su
</form>
```

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

An example to be validated

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

```
<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:

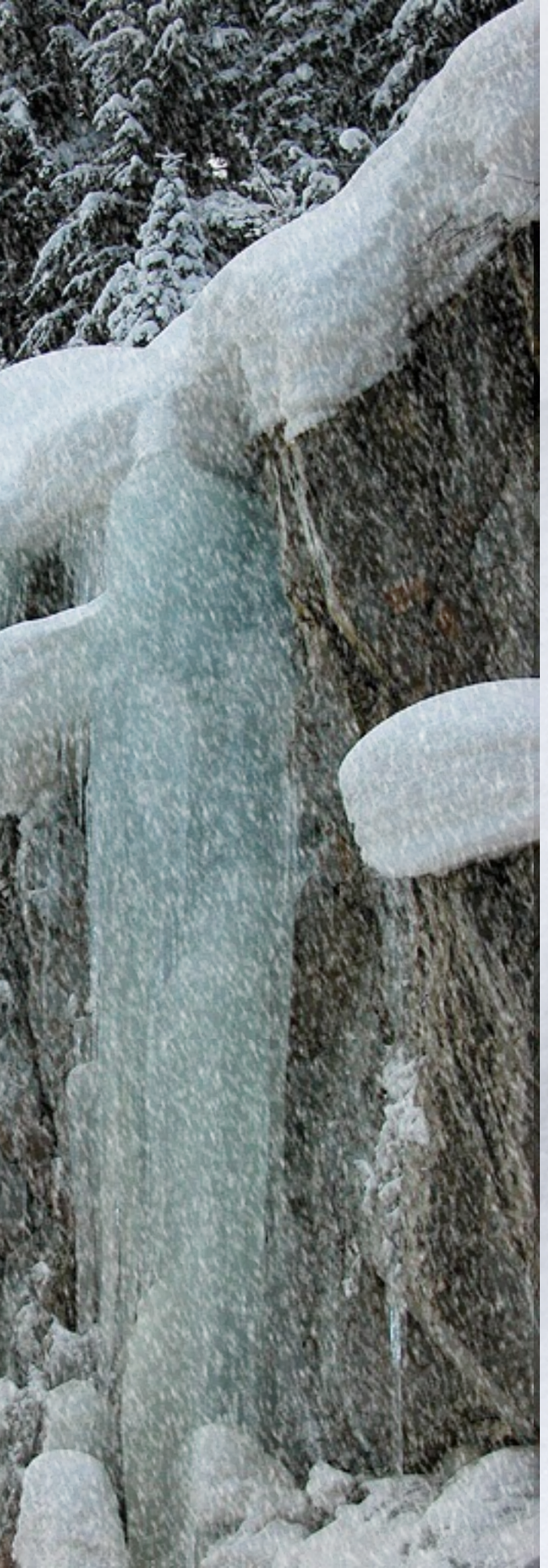
提交查询

output

Basic server-side validation code

- 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?)

```php
$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
}
```



Outline

- Submitting Data
- Processing Form data in PHP
- Form Validations
- Regular Expression
- Object-Oriented PHP

What is a regular expression?

- 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
  - supported by PHP, JavaScript, and other languages
  - many text editors (Notepad++, TextPad) allow regexes in search/replace

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

Basic regular expressions

- 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", ...

```
"/abc/"
```

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
 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", "Goooogle", ...
- ? 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)?

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 (\r, \t, \n, etc.); \S any non-whitespace

- What regular expression matches dollar amounts of at least $1000.00 ?

```
"/\$[1-9]\d{3,}\.\d{2}/"
```

Regular expressions in PHP

- 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 the given regex as the delimiter (similar to explode but more powerful)

Validate NAME, E-mail and URL

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

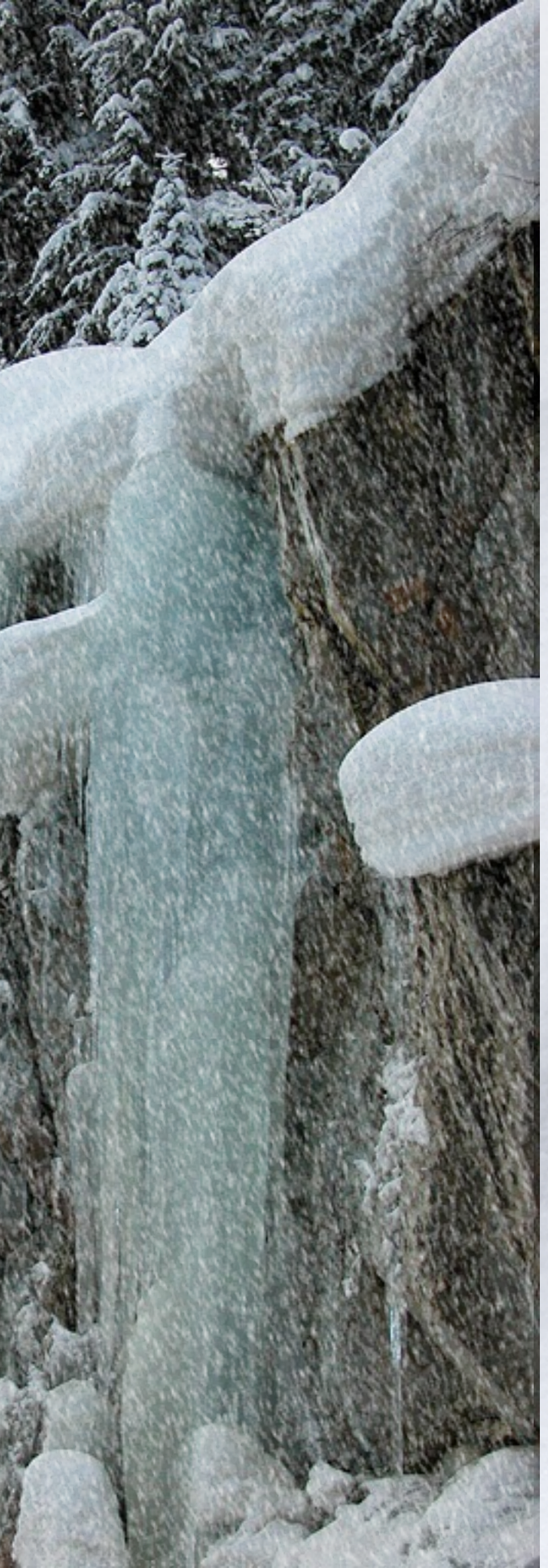
```
  $email = test_input($_POST["email"]);
  if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $emailErr = "Invalid email format";
  }
```

```
    $website = test_input($_POST["website"]);
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/
%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
```

Escape sequences

Single and double quoted PHP strings have special meaning of backslash. Thus if \ has to be matched with a regular expression \\, then "\\\\" or '\\\\' must be used in PHP code.

Perl或者PHP 的正则中，遇到元字符需要转义时，直接把元字符放在\Q\E之间即可。



Outline

- Submitting Data
- Processing Form data in PHP
- Form Validations
- Regular Expression
- Object-Oriented PHP

Why OOP?

- PHP is a primarily procedural language
- small programs are easily written without adding any classes or objects
- larger programs, however, become cluttered with so many disorganized functions
- grouping related data and behavior into objects helps manage size and complexity

Defining Class Properties and Methods

```php
<?php
 class MyClass
{
  public $prop1 = "I'm a class property!";
  public function setProperty($newval)
  {
      $this->prop1 = $newval;
  }
  public function getProperty()
  {
      return $this->prop1 . "<br />";
  }
}
$obj = new MyClass;
echo $obj->getProperty(); // Get the property value
$obj->setProperty("I'm a new property value!"); // Set a new one
echo $obj->getProperty(); // Read it out again to show the change
?>
```

Magic Methods in OOP

- To make the use of objects easier, PHP also provides a number of magic methods, or special methods that are called when certain common actions occur within objects. This allows developers to perform a number of useful tasks with relative ease.

Using Constructors and Destructors

```php
<?php
 class MyClass
{
  public $prop1 = "I'm a class property!";
  public function __construct()
  {
     echo 'The class "', __CLASS__, '" was initiated!<br />';
  }
  public function setProperty($newval)
  {
     $this->prop1 = $newval;
  }
  public function getProperty()
  {
     return $this->prop1 . "<br />";
  }
}
// Create a new object
$obj = new MyClass;
 // Get the value of $prop1
echo $obj->getProperty();
// Output a message at the end of the file
echo "End of file.<br />";
?>
```

Destructors

```
public function __destruct()
{
    echo 'The class "', __CLASS__, '" was destroyed.<br />';
}
```

- unset($obj);

Converting to a String

```
public function __toString()
 {
    echo "Using the toString method: ";
    return $this->getProperty();
 }
```

Using Class Inheritance

```
class MyOtherClass extends MyClass
{
  public function newMethod()
  {
    echo "From a new method in " . __CLASS__ . ".<br />";
  }
}
```

Assigning the Visibility of Properties and Methods

- Public Properties and Methods
  - All the methods and properties you've used so far have been public. This means that they can be accessed anywhere, both within the class and externally.
- Protected Properties and Methods
  - Protected Properties and Methods can only be accessed within the class itself or in descendant classes (classes that extend the class containing the protected method).
- Private Properties and Methods
  - A property or method declared private is accessible only from within the class that defines it. This means that even if a new class extends the class that defines a private property, that property or method will not be available at all within the child class.
- Static Properties and Methods
  - A method or property declared static can be accessed without first instantiating the class; you simply supply the class name, scope resolution operator, and the property or method name.
  - "One of the major benefits to using static properties is that they keep their stored values for the duration of the script."

Comparing Object-Oriented and Procedural Code

- Reason 1: Ease of Implementation
  - OOP will significantly reduce your workload if implemented properly
- Reason 2: Better Organization
  - easily packaged and cataloged
  - for example: All classes follow the naming convention class.classname.inc.php and reside in the inc folder of your application.

```
<?php
 function __autoload($class_name)
 {
    include_once 'inc/class.' . $class_name . '.inc.php';
 }
?>
```

- Reason 3: Easier Maintenance

Abstract classes and interfaces

- interfaces are supertypes that specify method headers without implementations
  - cannot be instantiated; cannot contain function bodies or fields
  - enables polymorphism between subtypes without sharing implementation code
- abstract classes are like interfaces, but you can specify fields, constructors, methods
  - also cannot be instantiated; enables polymorphism with sharing of implementation code

Reading materials

- PHP Regular Expression tutorials:
- http://www.phpro.org/tutorials/Introduction-to-PHP-Regex.html
- http://www.webcheatsheet.com/php/regular_expressions.php
- PHP Regular Expression examples: http://www.roscripts.com/PHP_regular_expressions_examples-136.html
- 精通正则表达式:第3版。(美)Jeffrey E.F.Friedl

Thanks!!!