PHP III

Super Globals

- PHP has a concept of super globals, these are global variables that are defined for you by PHP
  - There are always available in any scope, like an actual global variable
- The super global variables are (all are arrays)
  - `$GLOBALS` - all current global variables
  - `$_SERVER` - variables about the webserver or current script
  - `$_GET` - variables given by a GET request
  - `$_POST` - variables given by a POST request
  - `$_COOKIE` - HTTP cookies passed in the header
  - `$_FILES` - variables relating to file uploads over POST
  - `$_ENV` - Enviromental Variables
  - `$_REQUEST` - Like `$_GET`, `$_POST` and `$_COOKIE` all concatenated together

```php
<?php

function print_as_table($array) {
    print "<table><thead><tr><th>Key</th><th>Value</th></tr></thead><tbody>";
    if($array) {
      foreach(array_keys($array) as $key) {
        print "<tr><td>$key</td><td>$array[$key]</td></tr>";
      }
    } else {
      print '<tr><td colspan="2">None</td></tr>';
    }
    print "</tbody></table>";
}

?>
```

Available to view at https://www.csee.umbc.edu/~bwilk1/433/php_examples/web-variables.php

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Web Variables</title>
    <style>
      table { border-collapse: collapse; }
      td, th { border: 1px solid #333333; }
    </style>
  </head>
  <body>
    <h3>$_SERVER</h3>
    <?php print_as_table($_SERVER); ?>
    <h3>$_COOKIE</h3>
    <?php print_as_table($_COOKIE); ?>
    <h3>$_GET</h3>
    <?php print_as_table($_GET); ?>
    <h3>$_POST</h3>
    <?php print_as_table($_POST); ?>
    <h3>$_ENV</h3>
    <?php print_as_table($_ENV); ?>
```

An All in one Form Page

- Normal request to a website are done using GET
  - We can use this to our advantage to create a single page that both displays and processes a form
  - This is only used for simple forms
- The general syntax of this type of page is

```
<?php if($_SERVER['REQUEST_METHOD'] == 'GET') {?>
  <form method="post">
    #HTML for Form
  </form>
<?php } else {
  #Proecess form
}?>
```

Available at https://www.csee.umbc.edu/~bwilk1/433/php_examples/simple-form.php

```
<!DOCTYPE html>
<html lang="en">
<?php if($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
      <div>
        <label for="fahrenheit">Fahrenheit temperature:</label>
        <input type="text" name="fahrenheit" id="fahrenheit" />
        <input type="submit" value="Convert to Celcius" />
      </div>
    </form>
<?php } else {
      $fahrenheit = $_POST['fahrenheit'];
      $celcius = ($fahrenheit - 32) * 5 / 9;
      print "<p>$fahrenheit Fahrenheit is $celcius Celcius</p>";
    }
?>
  </body>
</html>
```

Multivalued Parameters

- PHP knows how to turn certain data into arrays automatically
  - The name of the input tag must end in `[]` for PHP to know to do this
  - Useful for checkboxes or multi-select

- A form sent using GET with the following HTML

```
<input type='checkbox' name='checks[]' value='hello'>Hello
<input type='checkbox' name='checks[]' value='hi'>Hi
<input type='checkbox' name='checks[]' value='howdy'>Howdy
```

- Creates an array in accesssible in the following way

```
$_GET['checks']
```

Available at https://www.csee.umbc.edu/~bwilk1/433/php_examples/checkboxes.php

```
<!DOCTYPE html>
<?php if($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
    <form action="<?php echo $_SERVER['_SELF'] ?>" method="post">
      <p>Select Languages you know</p>
      <p><input type="checkbox" name="languages[]" id="Perl" value="Perl" />
      <label for="Perl">Perl</label></p>
      <p><input type="checkbox" name="languages[]" id="JavaScript" value="JavaSc
ript" />
      <label for="JavaScript">JavaScript</label></p>
      <p><input type="checkbox" name="languages[]" id="PHP" value="PHP" />
      <label for="PHP">PHP</label></p>
      <p><input type="checkbox" name="languages[]" id="Python" value="Python" />
      <label for="Python">Python</label></p>
      <p><input type="submit" value="Submit" /></p>
    </form>
<?php } else {
        print '<p>You know:</p>';
        $languages = $_POST['languages'];
        print_r($languages);
    }
?>
```

File Uploads

- PHP has many functions to help you safely and easily handle file uploads
- Before we get a file to PHP to process, we have to set up the HTML form the correct way

```
<form enctype="multipart/form-data" action="some_url.php" method="POST">
  <input type="file" name="my_file">
</form>
```

  - The encoding type must be set for PHP to handle this correctly
  - File can only be sent over POST

File Uploads on the Server-Side

- Back in PHP, the file is uploaded to /tmp storage by default
  - This is good because /tmp as limited permissions so it is like a quarinteine zone
- For each file input you have, PHP creates an array in the `$_FILES` super global
- Import keys in that array
  - `$_FILES['my_file']['name']` - The original name
  - `$_FILES['my_file']['type']` - The MIME type as reported by the uploader, not checked, don't trust!!
  - `$_FILES['userfile']['tmp_name']` - The name on the server where the file was uploaded

Upload Safety

- Allowing uploads to a server opens you up to potential exploitation
- Steps you can take to reduce the risk
  - Never ever ever save uploaded files to a location accessible through a URL
  - Don't use the name given by the original uploader
    - Use a hash, or modify it in someway (You can keep a look up table somewhere if you need the original name too)
  - Validate the contents of the file
    - You can't trust the `type` key, but you can trust `finfo_file` function (PHP 5.3+)

```
<!DOCTYPE html>
<html>
    if($_SERVER['REQUEST_METHOD'] == 'GET') {
?>
    <form enctype="multipart/form-data" action="<?php $_SERVER['PHP_SELF'] ?>" m
ethod="post">
      <div>
        <label for="file">JPG File:</label>
        <input type="file" name="file" id="file"/>
        <input type="submit" value="Upload" />
      </div>
    </form>
<?php
    } else {
        // check for type - browser can lie, also check in PHP
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
        if($_FILES['file']['type'] != 'image/jpeg' || finfo_file($finfo, $_FILES
['file']['tmp_name']) != 'image/jpeg') {
            print '<p>File does not appear to be a JPG. ' .
                  '<a href="' . $_SERVER['PHP_SELF'] . '">Try Again</a></p>';
        // attempt to move to location
```

Sending HTTP Response Headers

- Related to processing forms, sometimes you want to someback something different than just HTML to the browser
- You can do this by changing the HTTP response headers
  - This must be done *before* any output is produced
- The PHP function `header` is used for this purpose
  - There are many different things you can send here, but a common one is to redirect to a different page

```php
<?php
header("Location: http://www.umbc.edu")
?>
```

Cache Control with Headers

- One common use of headers is to tell the browser and other network interfaces not to cache a page
- This can be set inside PHP by using the following code (from PHP Manual):

```php
<?php
 header("Cache-Control: no-cache, must-revalidate");
 header("Expires: Sat, 26 Jul 1997 05:00:00 GMT")
?>
```

Content Headers

- The `Content-Type` header tells the browser how to interperate the information it is receiving
  - This can be used to directly send images, pictures, pdfs, etc. and have the browser handle it correctly

```php
<?php
header('Content-Type: image/jpeg');
echo file_get_contents('myimage.jpg');
?>
```

Content-Header Examples

- Image: https://www.csee.umbc.edu/~bwilk1/433/php_examples/image.php

```php
<?php
  header("Content-Type: image/png");
  echo file_get_contents("maryland.png");
?>
```

Content-Header Examples

- PDF: https://www.csee.umbc.edu/~bwilk1/433/php_examples/pdf.php

```
<?php
    header('Content-Type: application/pdf');
    echo file_get_contents('https://about.umbc.edu/files/2017/09/2017-campus-ma
p.pdf');
?>
```

Content-Header Examples

- Video: https://www.csee.umbc.edu/~bwilk1/433/php_examples/video.php

```
<?php
    header("Content-Type: video/webm");
    echo file_get_contents("https://upload.wikimedia.org/wikipedia/commons/transcoded/2/26/Extraordinary_Adventures_of_Saturnino_Farandola_%281913%29.webm/Extraordinary_Adventures_of_Saturnino_Farandola_%281913%29.webm.360p.webm");

?>
```

Content-Header Examples

- PDF: https://www.csee.umbc.edu/~bwilk1/433/php_examples/no_header.php

```php
<?php
    echo file_get_contents('maryland.png');
?>
```

PHP Classes

- Since PHP 5, there has been support for true OOP style classes
  - The sytnax borrows heavily from Java strangely enough
    - But also from other scripting languages and C++

```
class className
{
    public/private $variableName

    public/private function functionName(){}
}
```

In [ ]:

```
class my_first_class{
    public $a_var;

    public function get_var(){
        return $this->a_var;
    }

}
```

In [ ]:

```
$obj = new my_first_class();
$obj->a_var = 10;
echo $obj->get_var();
```

Special Functions

- The following functions are called automatically in special contexts, and are all preceeded by two underscores
  - `__construct` (The object constructor)
  - `__destruct` (What is called when garbage collection is performed)
  - `__toString` (Used in print or echo)

In [ ]:

```
class Person{
    private $name;
    private $age;

    function __construct($name = "Jane Doe", $age = 30){
        $this->name = $name;
        $this->age = $age;
    }

    function __toString(){
        return $this->name . " is ". $this->age;
    }
}
```

In [ ]:

```
$p1 = new Person();
echo $p1;

$p2 = new Person("Timmy");
echo $p2;

$p3 = new Person("Harry", 45);
echo $p3;
```

Inheritance

- PHP permits only single inheritance, through use of the `extends` keyword
  - Have access to public and protected methods and members
  - To refer to parent method or member, used `parent` keyword
  - Use the `::` operator to access a method staticlly

In [ ]:

```
class Employee extends Person{
    private $workplace;

    function __construct($workplace = "UMBC"){
        parent::_construct();
        $this->workplace = $workplace;
    }

     function __construct($name="Joe Student", $age=19, $workplace = "UMBC"){
        parent::_construct($name,$age);
        $this->workplace = $workplace;
    }


    function __toString(){
        return parent::__toString . " and works at " . $this->workplace;
    }
}
```

Object Example

- Implement an object to represent a shopper (in an online store)

Object Practice

- Implement an object to represent a student

Interfaces

- Because PHP doesn't allow multiple inheritance (just like Java), they allow intrefaces (just like Java) that can be "implemented" (just like Java)
- The syntax for creating an interface is

```
interface name{
  public function name(); #Note there is no definition
}
```

Static and Constant

- Any method or member can be made static by adding the keyword `static` after the access modifier
  - A static method/member must be refered to using `::`
    - If you are inside the class defintion use `self::`
- A member can also be made constant by using the `const` keyword
  - These are static by default
  - They can only be made private in PHP 7.1+

In [ ]:

```
class my_static{
    private static $counter = 0;
    const UPDATE_BY = 1;

    function increment(){
        self::$counter += self::UPDATE_BY;
        return self::$counter;
    }
}
```

In [ ]:
```
$ms1 = new my_static();
echo $ms1->increment();
```

In [ ]:
```
$ms2 = new my_static();
echo $ms2->increment();
```

Sessions

- It many web applications it is useful to keep track of a user as the progress around your site
- PHP does this through a concept known as sessions, which requires cookies on the front end to use
  - The PHP server sends a cookie to the browser with a unique session id
  - Everytime it sees a particular session ID in a cookie, it populates `$_SESSION` with the appropriate values
- To use sessions, call the `session_start()` function in your PHP code

Available at https://www.csee.umbc.edu/~bwilk1/433/php_examples/session-store.php

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html lang="en">
<?php if($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
    <form action="<?php $_SERVER['PHP_SELF'] ?>" method="post">
      <div>
        <label for="name">Name:</label>
        <input type="text" name="name" id="name" />
        <input type="submit" value="Submit" />
      </div>
    </form>
<?php } else {
        $_SESSION['name'] = $_POST['name'];
        $name = $_POST['name'];
        print "<p>Hello, $name</p>";
    }
?>
  </body>
</html>
```

Available at https://www.csee.umbc.edu/~bwilk1/433/php_examples/session-recall.php

```php
<?php session_start(); ?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Recall Things from Session</title>
  </head>
  <body>
    <p>Hello, <?php echo $_SESSION['name'] ?></p>
  </body>
</html>
```

JSON

- JSON support was added to base PHP in PHP 5.2.0
- The two functions we care about are
  - `json_encode()` - Turns PHP object into JSON
  - `json_decode()` - Smartly produces an array/object/etc.

In [ ]:
```
$an_array = array(1,2,3,5,6);
echo json_encode($an_array);
```

In [ ]:
```
$an_object->date = "November 21, 2017";
$an_object->time = "12:??";
echo json_encode($an_object);
```

In [ ]:
```
$an_assoc_array = array('a' => 20, 'b' => 30);
echo json_encode($an_assoc_array);
```

In [ ]:
```
$p1 = new Person();
echo json_encode($p1);
```

In [ ]:
```
echo print_r(json_decode("[1,2,3,4,5]"),true);
```

In [ ]:
```
echo print_r(json_decode('{"name":"John Watson","dob":"November 10th 1860","birthp
lace":"London"}')
,true);
```

JSON and PHP Object Comparison

- When going from PHP to JSON
  - A PHP Object is always a JSON object
  - A PHP array is an JSON object when it is associative
  - A PHP array is a JSON array when it has no keys other than continuous integers
- When going from JSON to PHP
  - A JSON object is always a generic PHP object of type Object
  - A JSON array is always a simple indexed array