Zend Certification Exam Preparation

What This Course Is Not

- Coherent coverage of all topics
- A guarantee of success
- In-depth
- Accredited by Zend

Aims of Today

- FAST overview of certification content
- Refresher/Reminder on well-known things
- Comments on exam styles
- Identification of any weak points
- Provision of resources for further study

Meet the PHP Manual

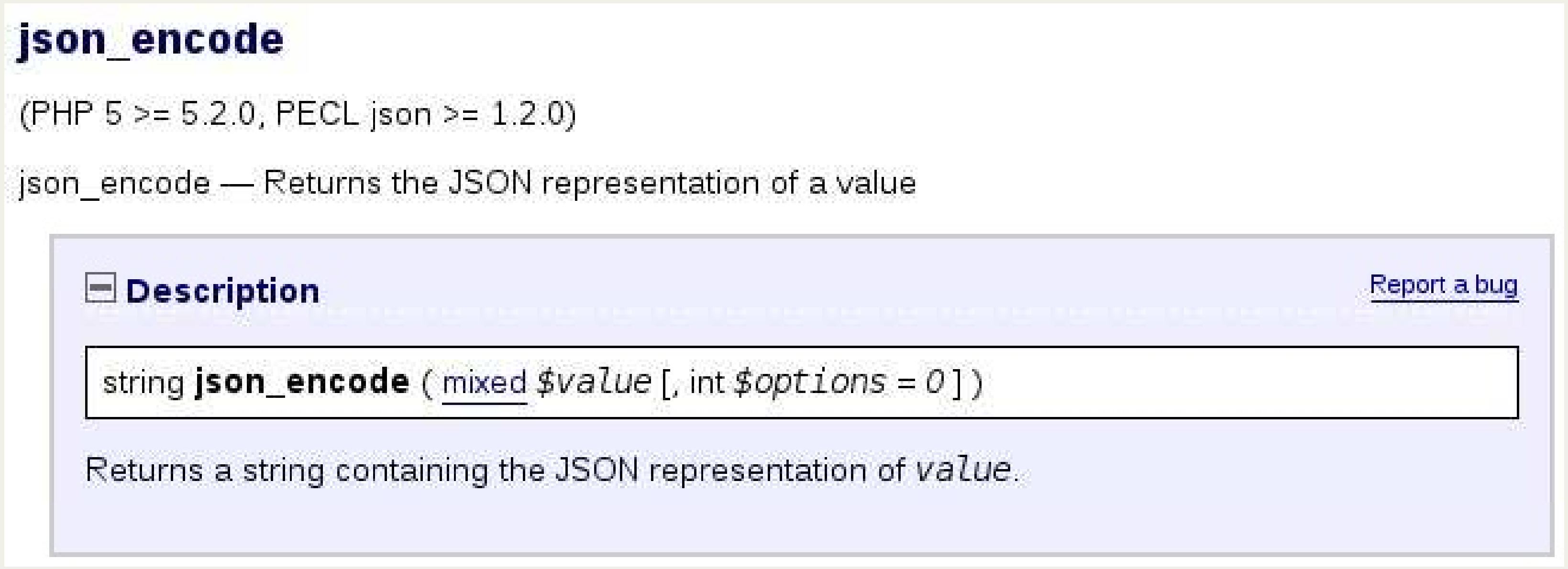
Most Important Resource: php.net

- Main Page: `http://php.net`
  - Local versions: `http://uk2.php.net`
  - Many translations available
- Cool Shortcut: **`http://php.net/[function name]`**
  - Redirects you straight to that page
  - `http://php.net/array_walk`
  - `http://uk2.php.net/manual/en/function.array-walk.php`

Anatomy of a PHP Manual Page

- Description
- Parameters
- Return Values
- Changelog
- Examples
- See Also
- User-Contributed Notes

Description



json_encode

(PHP 5 >= 5.2.0, PECL json >= 1.2.0)

json_encode — Returns the JSON representation of a value

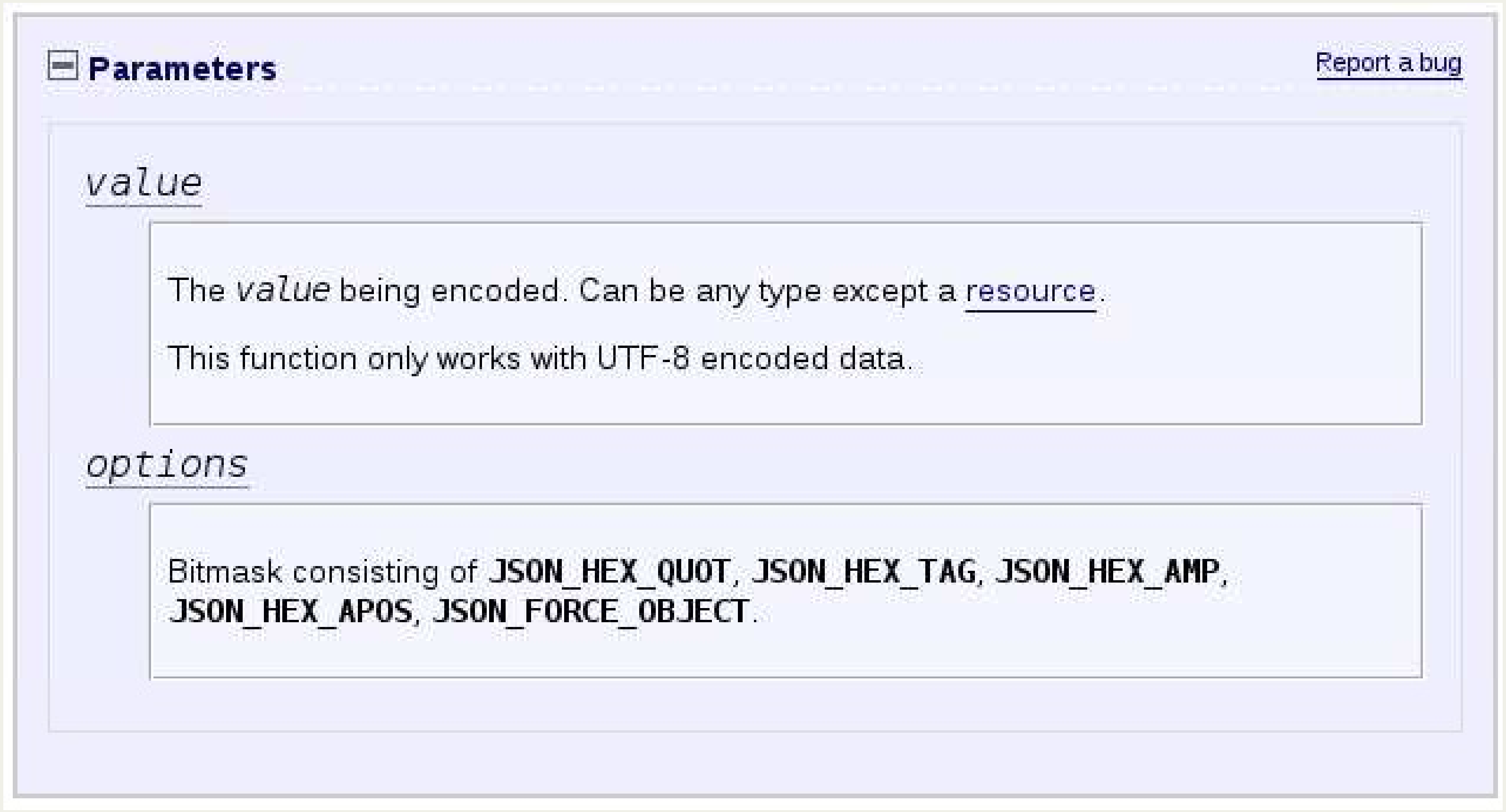
Description

Report a bug

string **json_encode** (mixed *$value* [, int *$options* = *0*])

Returns a string containing the JSON representation of *value*.

Parameters


Parameters

Report a bug

value

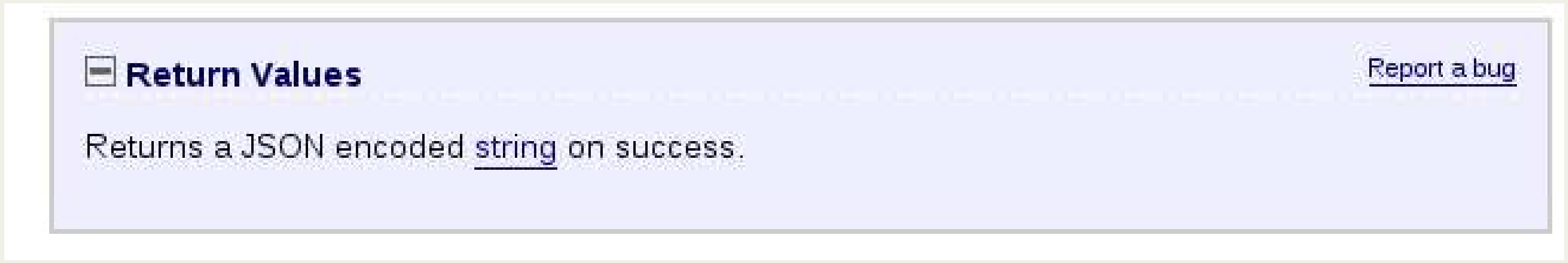
The *value* being encoded. Can be any type except a resource.

This function only works with UTF-8 encoded data.

options

Bitmask consisting of **JSON_HEX_QUOT**, **JSON_HEX_TAG**, **JSON_HEX_AMP**, **JSON_HEX_APOS**, **JSON_FORCE_OBJECT**.

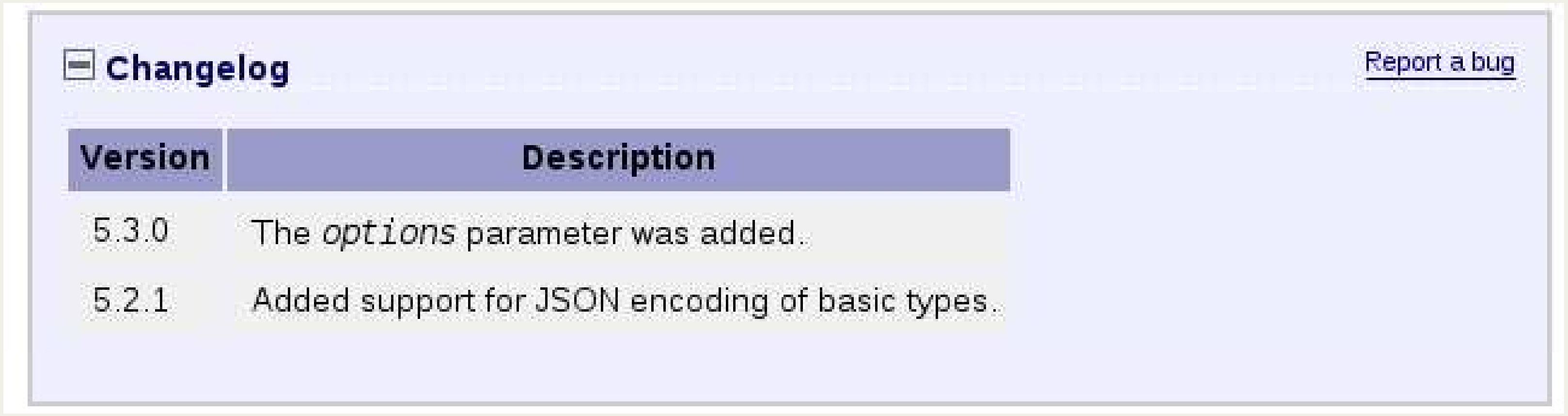
Return Values


Return Values

Report a bug

Returns a JSON encoded string on success.

Changelog

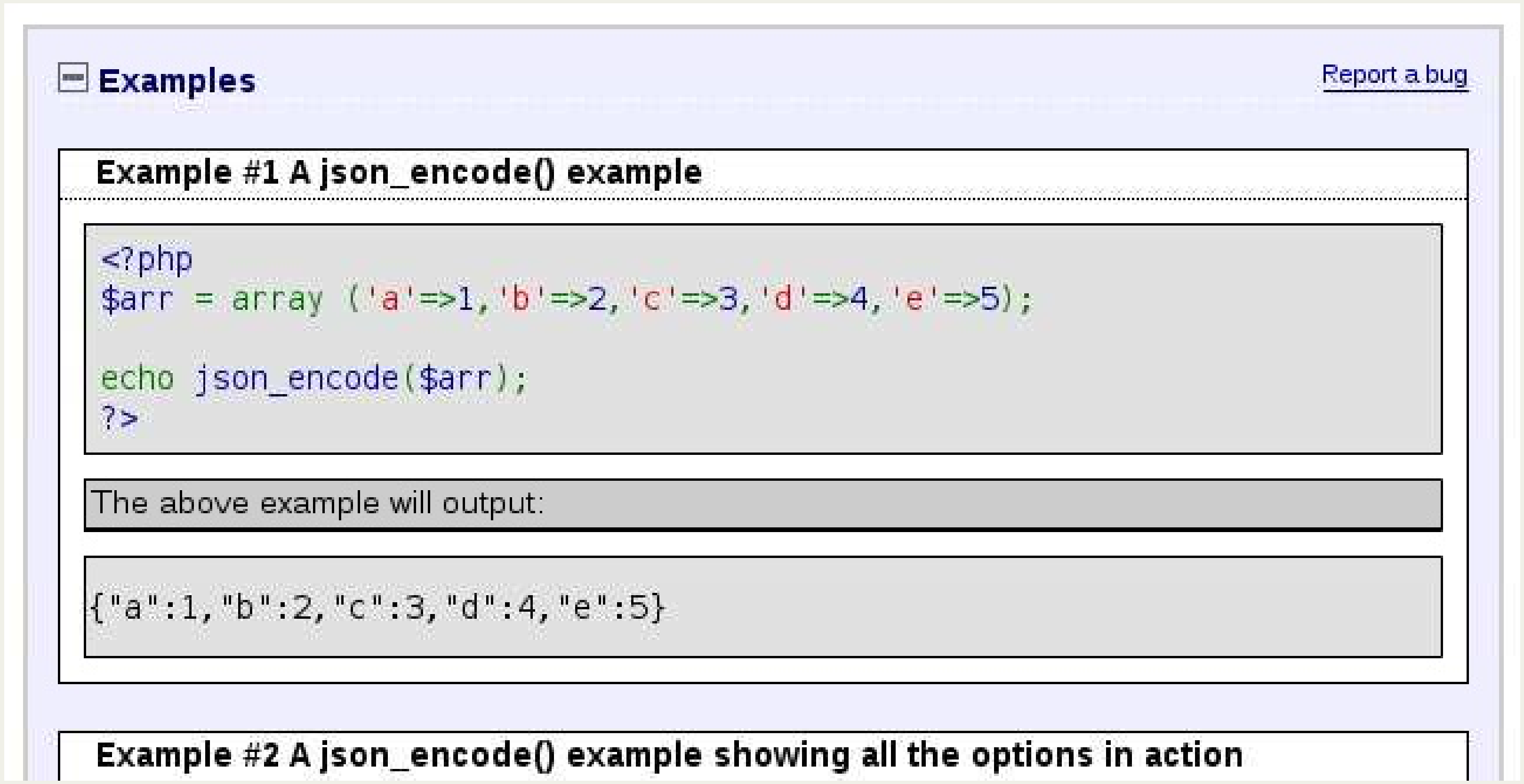


Changelog

Report a bug

Version	Description
5.3.0	The *options* parameter was added.
5.2.1	Added support for JSON encoding of basic types.

Examples



Examples Report a bug

Example #1 A json_encode() example

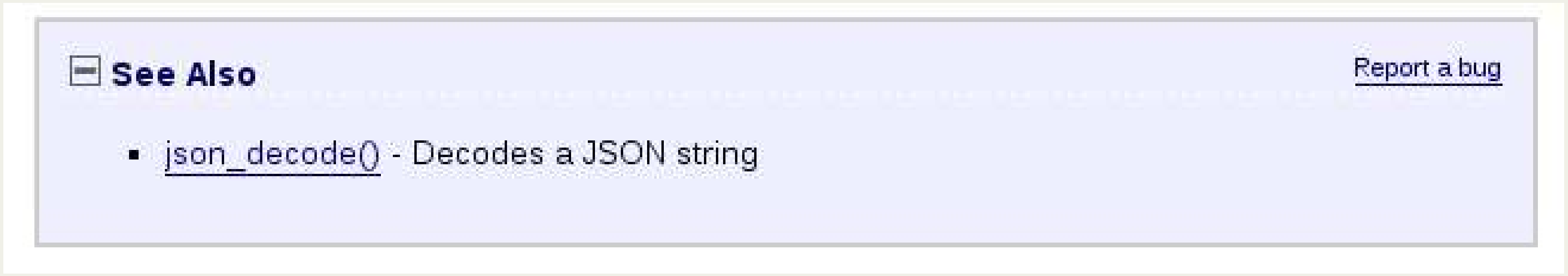
```
<?php
$arr = array ('a'=>1,'b'=>2,'c'=>3,'d'=>4,'e'=>5);

echo json_encode($arr);
?>
```

The above example will output:

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

Example #2 A json_encode() example showing all the options in action

See Also

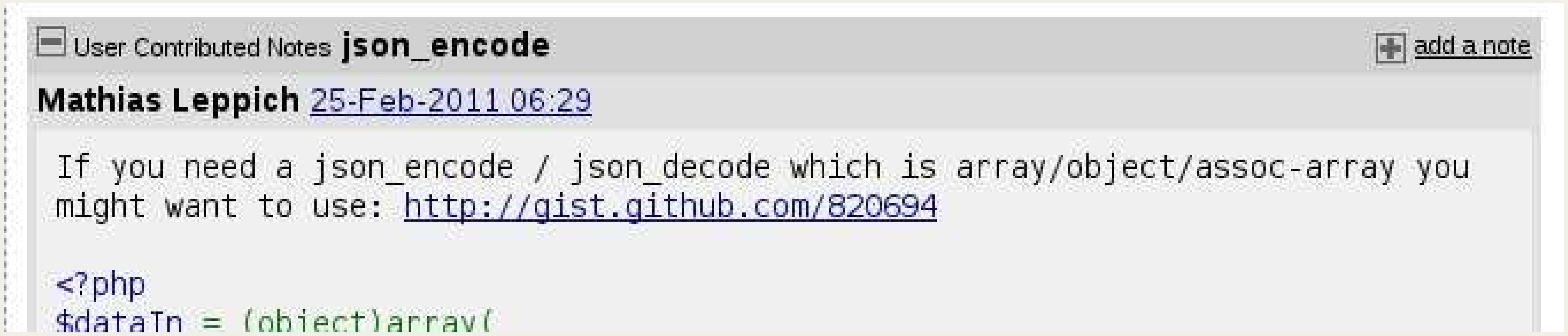


See Also

Report a bug

- json_decode() - Decodes a JSON string

User-Contributed Notes


User Contributed Notes **json_encode** add a note

Mathias Leppich 25-Feb-2011 06:29

```
If you need a json_encode / json_decode which is array/object/assoc-array you
might want to use: http://gist.github.com/820694

<?php
$dataIn = (object)array(
```

Boring Basics

PHP Tags

Many different ways to open PHP tags:

- Standard Tags: **`<?php`** and **`?>`**
- Short Tags: **`<?`** and **`?>`**
- Script Tags: **`<script language="php">`** and **`</script>`**
- ASP-Style Tags: **<%** and **%>**

Only the first is recommended. Others are enabled with ini settings

- **`asp_tags`**
- **`short_open_tag`**

Commenting Code

Many different ways to do this too!

```
// a one-line comment

# a less common format of one-line comment

/* A comment
which can span
a great many
lines */

/**
 * More common multi-line commenting
 *
 * @param string The variable for the method
 */
```

See Also: PHPDocumentor

- http://www.phpdoc.org/

Operators

- Operators are how we tell PHP what to do with our variables
- ZCE expects you to know many different types - look out for precedence to trip you up
- See Also: `http://php.net/manual/en/language.operators.php`

Arithmetic Operators

-$a	Negation
$a + $b	Addition
$a - $b	Subtraction
$a * $b	Multiplication
$a / $b	Division
$a % $b	Modulus

Example of modulus operator:

```
echo (17 % 4); // 1
echo (13 % 5); // 3
```

Shorthand Operators

A contraction of operating on something and assigning the result

- `$a = $a + $b`

becomes:

- `$a += $b`

The same thing works for `-` `/` `*` `%` and `.`

Ternary Operator

This is a shortcut for an if/else statement.

```php
$a = isset($_GET['param1']) ? $_GET['param1'] : 10;
```

There's a contraction of it too, where the first two items match:

```php
$pages = $_GET['pages'] ? $_GET['pages'] : 20;
$pages = $_GET['pages'] ?: 20;
```

Comparison Operators

A great source of trick questions!

`==`	Equal
`===`	Strictly equal
`!=`	Not equal
`!==`	Strictly not equal

Comparisons: The strpos Trap

```
$tagline = "PHP Made Easy";

if(strpos(strtolower($tagline), 'php')) {
    echo 'php tagline: ' . $tagline;
}
```

Comparisons: The strpos Trap

```php
$tagline = "PHP Made Easy";

if(false !==  strpos(strtolower($tagline), 'php')) {
    echo 'php tagline: ' . $tagline;
}
```

Data Types

PHP is dynamically weakly typed. It does "type juggling" when data types don’t match.

PHP data types:

- integer
- float
- boolean
- string
- array
- object
- resource

Number Systems

System	Characters	Notes
Binary	01	used in logical calculations
Decimal	0123456789	"normal" numbers
Octal	01234567	written with a leading 0
Hex	0123456789abcdef	used for HTML colours

`http://www.lornajane.net/posts/2011/Number-System-Primer`

Variables

- Start with a **`$`**
- Don’t need to be initialised
- Represent a value, of any type
- Start with a letter, then can contain letters, numbers and underscores
- Are usually lowercase or CamelCase

Variable variables

```
$name = "Fiona";
$var = "name";

echo $$var;  // Fiona
```

Constants

- Represent a value, of any type
- Are initialised with **`define()`** and cannot change
- Do not have a **`$`** in front of their name
- Start with a letter, then can contain letters, numbers and underscores
- Are usually UPPER CASE

Control Structures

We’ll look at examples of each of:

- if/elseif/else
- switch
- for
- while
- do..while
- foreach

If/ElseIf/Else

```php
if($hour < 10) {
    $beverage = "coffee";
} elseif($hour > 22) {
    $beverage = "hot chocolate";
} else {
    $beverage = "tea";
}
```

Switch

```
switch(date('D')) {
    case 'Monday':
                    echo "Back to work";
                    break;
    case 'Friday':
                    echo "Almost the weekend!";
                    break;
    case 'Saturday':
    case 'Sunday':
                    echo "Not a working day :)";
                    break;
    default:
                    echo "Just another day";
                    break;
}
```

For

```php
for($i=10; $i > 0; $i--) {
    echo $i . " green bottles, hanging on the wall\n";
}
```

While

```
$finished = false;
while(!$finished) {
    $second = substr(date('s'), 1);
    if($second == '7') {
        $finished = true;
    } else {
        echo $second;
    }
    sleep(3);
}
```

Do .. While

```
$finished = false;
do {
    $second = substr(date('s'), 1);
    if($second == '7') {
        $finished = true;
    } else {
        echo $second;
    }
    sleep(3);
} while(!$finished);
```

Foreach

```

$list = array(
    "chicken",
    "lamb",
    "reindeer");

foreach($list as $value) {
    echo "On the menu: " . $value . "\n";
}
```

Foreach

```
$list = array(
    "chicken",
    "lamb",
    "reindeer");

// make plural
foreach($list as $key => $value) {
    $list[$key] = $value . "s";

}

foreach($list as $value) {
    echo "On the menu: " . $value . "\n";
}
```

Break and Continue

- **break** go to after the next **}**
- **continue** go to the end of this iteration

Both can have a number to allow them to operate on nested structures

Strings and Patterns

100 String Functions

Anyone want to talk about all hundred string functions?

String Functions

Anything you want to do with a string, there's a function for that

Special terminology

- **needle**: the thing you are looking for
- **haystack**: the place you are looking

Quotes

- Single quote ′
  - contains a string to be used as it is
- Double quote "
  - can contain items to evaluate
  - you can use (simple) variables here

Escape Characters

Escape character is backslash `\`. Useful when you want to:

- put a $ in a string
- use a quote in a quoted string
- disable any special character meaning

We sometimes need to escape the escape character

```
echo "Escape character is backslash \\";
```

Formatting Strings

Often we'll concatenate as we need to, but we can also use formatting functions

```
$animals = array(
    array("animal" => "cat", "legs" => 4),
    array("animal" => "bee", "legs" => 6),
    array("animal" => "peacock", "legs" => 2));

foreach($animals as $animal) {
    printf("This %s has %d legs\n",
        $animal['animal'], $animal['legs']);
}
```

See also: `*printf() and *scanf()`

HEREDOC

Ask PHP to output everything until the placeholder

```php

$item = "star";
echo <<<ABC
Star light, $item bright,
    The first $item I see tonight;
I wish I may, I wish I might,
    Have the wish I wish tonight
ABC;
```

NOWDOC

```
echo <<<'ABC'
Star light, star bright,
    The first star I see tonight;
I wish I may, I wish I might,
    Have the wish I wish tonight
ABC;
```

Regular Expressions

- Often abbreviated to "RegEx"
- Describe a pattern for strings to match

```
/b[aeiou]t/
```

Matches "bat", "bet", "bit", "bot" and "but"

Regular Expressions

- Often abbreviated to "RegEx"
- Describe a pattern for strings to match

```
/b[aeiou]t/
```

Matches "bat", "bet", "bit", "bot" and "but"

Also matches "cricket bat", "bitter lemon"

Using Regex in PHP

```php
$pattern = '/b[aeiou]t/';
// returns the number of times there's a match
echo preg_match($pattern, "bat"); // 1
```

Many other string handling functions use regex, and there’s also **`preg_match_all`**

Character Ranges

We can use ranges of characters, e.g. to match hex:

```
/[0-9a-f]*/
```

Character Ranges

We can use ranges of characters, e.g. to match hex:
`/[0-9a-f]*/`

Upper and lower case are distinct; for alphanumeric: **`/[0-9a-zA-Z]/`**

Character Ranges

We can use ranges of characters, e.g. to match hex:
`/[0-9a-f]*/`

Upper and lower case are distinct; for alphanumeric: **`/[0-9a-zA-Z]/`**

If you want to allow another couple of characters, go for it:
`/[0-9a-zA-Z_]/`

Character Ranges

We can use ranges of characters, e.g. to match hex:
`/[0-9a-f]*/`

Upper and lower case are distinct; for alphanumeric: **`/[0-9a-zA-Z]/`**

If you want to allow another couple of characters, go for it:
`/[0-9a-zA-Z_]/`

To match any character, use a dot .

Character Classes

There are preset ways of saying "number", "whitespace" and so on:

`\w`	word character
`\s`	whitespace
`\d`	digit

When used in uppercase, these are negated

Pattern Modifiers

We can add modifiers to our pattern, to say how many matching characters are allowed.

`?`	0 or 1 time
`*`	0 or more times
`+`	1 or more times
`{n}`	n times
`{n,}`	n or more times
`{n,m}`	between n and m times
`{,m}`	up to m times

`/b[aeiou]*t/`
Matches "bat" and "bit" etc, but also "boot" and "boat"

Anchoring Patterns

To stop us from matching "cricket bat", we can anchor

`^`	start of line
`$`	end of line
`\A`	start of string
`\Z`	end of string

`/^b[aeiou]t/` Will match "battering ram" but not "cricket bat"

Regex Delimiters

- Regexes often contained by a `/`
- Messy if your expression also contains slashes (e.g. for a URL)
- Also common to use pipe or hash
- Any matching pair works

Regex Resources

Brain exploding? Use this cheat sheet from addedbytes.com@

http://bit.ly/kiWlbZ

Arrays

Array Syntax

Some examples of the syntax around arrays:

```

$items = array("pen", "pencil", "ruler");
$items[7] = "calculator";
$items[] = "post-its";

var_dump($items);
```

Outputs this:

```
Array
(
    [0] => pen
    [1] => pencil
    [2] => ruler
    [7] => calculator
    [8] => post-its
)
```

This is an **enumerated** array

Associative Arrays

Associative arrays have named keys

```

$characters[] = array("name" => "Lala",
    "colour" => "Yellow");
$characters[] = array("name" => "Tinky Winky",
    "colour" => "Purple");
```

This is a nested associative array

```
Array
(
    [0] => Array
        (
            [name] => Lala
            [colour] => Yellow
        )

    [1] => Array
        (
            [name] => Tinky Winky
            [colour] => Purple
        )
```

Array Functions

Only 75+ of these

(12 just for sorting)

Functions

Functions

Declaring functions:

```php
function addStars($message) {
    return '** ' . $message . ' **';
}
```

Calling functions:

```php
echo addStars("twinkle");
```

Functions and Arguments

Passing many arguments:

```php

function setColour($red, $green, $blue) {
    return '#' . $red . $green . $blue;
}

echo setColour('99','00','cc'); //#9900cc
```

And optional ones:

```php

function setColourAndIntensity($red, $green, $blue,
        $intensity = 'ff') {
    return '#' . $red . $green . $blue . $intensity;
}
echo setColourAndIntensity('99','00','cc'); //#9900ccff
echo setColourAndIntensity('99','00','cc', '66'); //#9900cc66
```

Optional arguments should be the last on the list

Return Values

- By default, functions return NULL
- Good practice to return values
- **Check** if value is returned or assigned

Return Values

- By default, functions return NULL
- Good practice to return values
- **Check** if value is returned or assigned
- **Now check again**

Functions and Scope

- Functions are a "clean sheet" for variables
- Outside values are not available
- Pass in as parameters to use them
- There is also a **`global`** keyword
    - it was acceptable at one time
    - now considered poor practice

Scope Examples

```

function doStuff() {

    $apples++;
}

$apples = 4;
echo $apples; //4
doStuff();
echo $apples; //4
```

Scope Examples

```
function doStuff() {
    global $apples;
    $apples++;
}

$apples = 4;
echo $apples; //4
doStuff();
echo $apples; //5
```

Pass by Reference

By default:

- Primitive types are copies
- Objects are references

To pass a variable by reference, declare it in the function with &:

```
function moreCakes(&$basket) {
    $basket++;
    return true;
}

$basket = 0;
moreCakes($basket);
moreCakes($basket);
echo $basket; // 2
```

Call-Time Pass-By-Reference

- The & goes in the function declaration
- **NOT** in the call
- PHP 5.3 gives an error about call-time pass-by-reference

See also:
`http://php.net/manual/en/language.references.pass.php`

Anonymous Functions

- Literally functions with no name
- More convenient than `create_function()`
- Called **lambdas**
- Unless they use variables from the outside scope
- Then they are called **closures**

Great explanation: `http://bit.ly/kn9Arg`

Lambda Example

```php

$ping = function() {
    echo "ping!";
};

$ping();
```

Closure Example

```php

$message = "hello";
$greet = function ($name) use ($message) {
    echo $message . ' ' . $name;
};

$greet('Daisy'); // hello Daisy
```

Closure Example

```php
$message = "hello";
$greet = function ($name) use ($message) {
    echo $message . ' ' . $name;
};
$message = "hey";
$greet('Daisy'); // hello Daisy
```

Namespaced Functions

Namespaces are a 5.3 feature

- Avoid naming collision
- Avoid stupid long function names

```
namespace lolcode;

function catSays() {
    echo "meow";
}
```

```
lolcode\catSays();
```

http://blogs.sitepoint.com/php-53-namespaces-basics/

Files, Streams and other Fun

Working with Files

There are two main ways to work with files

- All at once, using **`file_*`** functions
- In bite-sized pieces, using **`f*`** functions

Working with Files

There are two main ways to work with files

- All at once, using **`file_*`** functions
- In bite-sized pieces, using **`f*`** functions

For platform independence we have **`DIRECTORY_SEPARATOR`**

File Functions

Read and write files using **`file_get_contents()`** and **`file_put_contents()`**

```
$words = file_get_contents('words.txt');
echo $words; // This is a file containing words.
file_put_contents('words.txt', str_replace('words', 'nonsense', $words
```

The f* Functions

- Use a file handle from **`fopen()`**
- Read in chunks, using **`fgets()`**
- Or all in one go, using **`file()`** or **`fread()`**
- Write with **`fwrite()`**
- Close handle with **`fclose()`**

Fopen

Fopen can operate in various modes, passed in as the 2nd argument

r	For reading
w	for writing, empties the file first
a	for writing, adding onto the end of the file
x	for writing, fail if the file exists
c	for writing, start at the top
+	in combination with any of the above, to enable reading/writing also
b	binary mode

Reading from Files

```php

$fh = fopen('lorem.txt', 'r');
while(!feof($fh)) {
    echo fgets($fh);
}

flcose($fh);
```

Notice **`feof()`** which returns true when we reach the end of the file

Writing to Files

```php

$fh = fopen('polly.txt', 'w');

for($i=0; $i<3; $i++) {
    fwrite($fh, 'Polly put the kettle on' . PHP_EOL);
}
fwrite($fh, 'We\'ll all have tea' . PHP_EOL);
```

File System Functions

Other useful file and directory functions

- `glob()`
- `is_dir()`
- `is_file()`
- `copy()`
- `rename()`
- `unlink()`

PHP Setup and Configuration

phpinfo()

Call this function to find out:

- What version of PHP you have
- Which **`php.ini`** is being used
- What your config settings are
- Which extensions are installed

Common Config Settings

- `error_reporting`
- `display_errors`
- `memory_limit`
- `post_max_size`
- `include_path`
- `file_uploads`
- `upload_max_filesize`

http://php.net/manual/en/ini.core.php

PHP include_path

- Use **`get_include_path()`** to get current
- There is a PATH_SEPARATOR for platform independence
- Set with **`set_include_path()`**

Include paths can be useful for libraries, etc

Question Styles

Question Types

- Multiple choice
  - pick one answer
  - may include "none of the above"
- Multiple choice, multiple option
  - checkboxes rather than radio buttons
  - if you tick too few, the software will tell you
- Free text
  - function name, script output, or other string

Sample Question

What is the output of the following code?

```
<code>
echo strlen(sha1('0'), true);
</code>
```

(textarea)

Sample Question

What does the **`max_file_uploads`** configuration option contain?

- **A** The maximum number of file uploads per session
- **B** The maximum number of file uploads per request
- **C** The maximum number of file uploads per user
- **D** The maximum number of file uploads before the web service process is restarted

Sample Question

What will the following code print?

```php
$str = printf('%.1f',5.3);
echo 'Zend PHP Certification ';
echo $str;
```

- **A** Zend Certification 5.3
- **B** Zend PHP Certification
- **C** 5.3Zend PHP Certification 3

Sample Question

What is the output of the following code?

```
$a = 1;
++$a;
$a *= $a;
echo $a--;
```

- **A** 4
- **B** 3
- **C** 5
- **D** 0
- **E** 1

Sample Question

Which of the following statements about static functions is true?

- **A** Static functions can only access static properties of the class
- **B** Static functions cannot be called from non-static functions
- **C** Static functions cannot be abstract
- **D** Static functions cannot be inherited

Sample Question

```
class A {
    protected $a = 1;
    function x() { echo $this->a++; }
}

$a = new A();
$b = $a;
$c = new A();
$b->x();
$a->x();
$c->x();
$b = $c;
$b->x();
$a->x();
```

- **A** 11122
- **B** 12345
- **C** 12123
- **D** 12134

Object Orientation

Classes and Objects

A class is a recipe for making an object

```
class Robot {
    public $name;

    public function flashLights($pattern) {
        // look!  Pretty flashing lights
        return true;
    }
}
```

An object is an instance of a class

```
$marvin = new Robot();
```

Object Methods and Properties

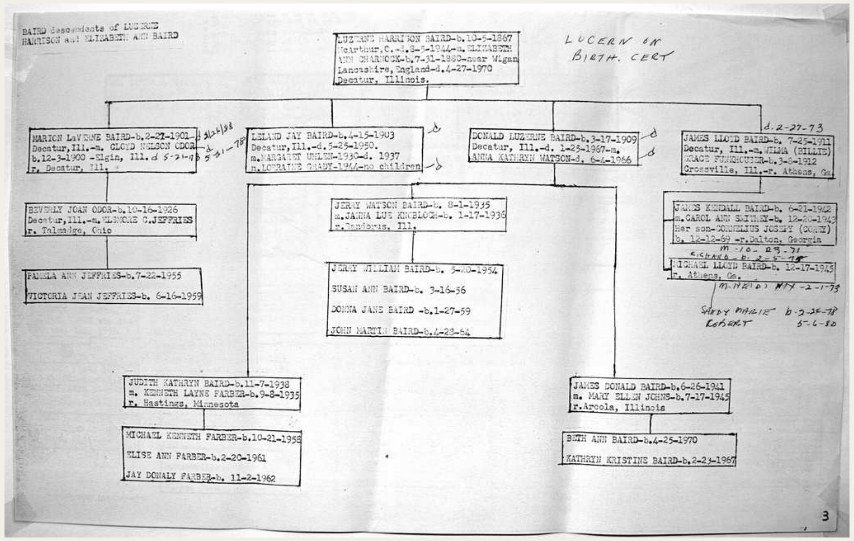
Object variables are "properties" and their functions are "methods"

```
$marvin = new Robot();
$marvin->name = 'Marvin';
$marvin->flashLights();
```

Inheritance

OOP supports inheritance

- similar classes can share a parent and override features
- improves modularity, avoids duplication
- classes can only have one parent (unlike some other languages)
- classes can have many children
- there can be as many generations of inheritance as we need



Inheritance Examples

```php
class Table {

    public $legs;

    public function getLegCount() {
        return $this->legs;
    }
}

class DiningTable extends Table {}
```

```php
$newtable = new DiningTable();
$newtable->legs = 6;
echo $newtable->getLegCount(); // 6
```

Visibility

We can control which parts of a class are available and where:

- **`public`**: always available, everywhere
- **`private`**: only available inside this class
- **`protected`**: only available inside this class and descendants

This applies to both methods and properties

Protected Properties

```
class Table {
    protected $legs;

    public function getLegCount() {
        return $this->legs;
    }

    public function setLegCount($legs) {
        $this->legs = $legs;
        return true;
    }
}
```

```
$table = new Table();
$table->legs = 4;

// Fatal error: Cannot access protected property Table::$legs in /.../
```

Protected Properties

```php
class Table {
    protected $legs;

    public function getLegCount() {
        return $this->legs;
    }

    public function setLegCount($legs) {
        $this->legs = $legs;
        return true;
    }
}
```

```php
$table = new Table();
$table->setLegCount(4);

echo $table->getLegCount();
```

Protected Methods

Access modifiers for methods work exactly the same way:

```
class Table {
    protected function getColours() {
        return array("beech", "birch", "mahogany");
    }
}
```

```
class DiningTable extends Table {
    public function colourChoice() {
        return parent::getColours();
    }
}
```

If **`Table::getColours()`** were private, DiningTable would think that method was undefined

Object Keywords

- **`parent`**: the class this class extends

Object Keywords

- **`parent`**: the class this class extends
- **`self`**: this class, usually used in a static context, instead of **`$this`**
    - **WARNING:** in extending classes, this resolves to where it was declared
    - This was fixed in PHP 5.3 by "late static binding"

Object Keywords

- **`parent`**: the class this class extends
- **`self`**: this class, usually used in a static context, instead of **`$this`**
    - **WARNING:** in extending classes, this resolves to where it was declared
    - This was fixed in PHP 5.3 by "late static binding"
- **`static`**: the class in which the code is being used
    - Just like **`self`** but actually works :)
    - Added in 5.3 "Late Static Binding"

Comparing Objects

- Comparison ==
    - objects must be of the (exact) same class
    - objects must have identical properties
- Strict comparison ===
    - both arguments must refer to the same object

Static Methods

We can call methods without instantiating a class

- `$this` is not available in a static method
- use the `::` notation (paamayim nekudotayim)
- used where we don’t need object properties

```
class Table {
    public static function getColours() {
        return array("beech", "birch", "mahogany");
    }
}
```

```
$choices = Table::getColours();
```

Static Properties

- Exactly like static methods
- Use **`static`** when declaring them
- Can be accessed without instantiating the class

Example: Singleton

```
class Singleton
{
  private static $classInstance;

  private function __construct () {}

  static function getInstance () {
    if (! isset(self::$classInstance)) {
      self::$classInstance = new Singleton();
    }
    return (self::$classInstance);
  }
}
```

Class Constants

- Class constants are similar to static properties
- But constants can’t change

```
class Robot {
    const MESSAGE = "Here I am, brain the size of a planet";
    public $name;

    public function flashLights($pattern) {
        // look!  Pretty flashing lights
        return true;
    }
}

echo Robot::MESSAGE;
```

Interfaces

- prototypes of class methods
- classes "implement" an interface
- they must implement all these methods
- the object equivalent of a contract

PHP does not have multiple inheritance

Example Interface: Countable

This interface is defined in SPL, and it looks like this:

```
Interface Countable {
    public function count();
}
```

RTFM: `http://uk2.php.net/manual/en/class.countable.php`

Autoloading

Use **`include`** and **`require`** to bring class code into our applications.

We can also use **autoloading** if our classes are predictably named.

```
function __autoload($classname) {

        if(preg_match('/[a-zA-Z]+Controller$/',$classname)) {
                include('../controllers/' . $classname . '.php');
                return true;
        } elseif(preg_match('/[a-zA-Z]+Model$/',$classname)) {
                include('../models/' . $classname . '.php');
                return true;
        } elseif(preg_match('/[a-zA-Z]+View$/',$classname)) {
                include('../views/' . $classname . '.php');
                return true;
        }
}
```

No need to include/require if you have autoloading

The instanceOf Operator

To check whether an object is of a particular class, use **`instanceOf`**

```
$table = new DiningTable();

if($table instanceOf DiningTable) {
    echo "a dining table\n";
}

if($table instanceOf Table) {
    echo "a table\n";
}
```

InstanceOf will return true if the object:

- is of that class
- is of a child of that class
- implements that interface

Type Hinting

We have type hinting in PHP for complex types. So we can do:

```

function interrogate(Robot $robot) {
    // imagine something more exciting
    while($robot->getStatus() ==  'OK') {
        askAnotherQuestion($robot);
    }
    return true;
}
```

PHP will error unless the argument:

- is of that class
- is of a child of that class
- implements that class

Raising Exceptions

In PHP, we can throw any exception, any time.

```php
function addTwoNumbers($a, $b) {
    if(($a == 0) || ($b == 0)) {
        throw new Exception("Zero is Boring!");
    }

    return $a + $b;
}

echo addTwoNumbers(3,2); // 5
echo addTwoNumbers(5,0); // error!!
```

```
Fatal error: Uncaught exception ’Exception’ with message ’Zero is Boring!’ in /
Stack trace:
#0 /.../exception.php(12): addTwoNumbers(5, 0)
#1 {main}
    thrown in /.../exception.php on line 5
```

Extending Exceptions

We can extend the Exception class for our own use

```
class DontBeDaftException extends Exception {
}

function tableColour($colour) {
    if($colour == "orange" || $colour == "spotty") {
        throw new DontBeDaftException($colour . 'is not acceptable');
    }
    echo "The table is $colour\n";
}

try {
    tableColour("blue");
    tableColour("orange");
} catch (DontBeDaftException $e) {
    echo "Don't be daft! " . $e->getMessage();
} catch (Exception $e) {
    echo "The sky is falling in! " . $e->getMessage();
}
```

Magic Methods

In PHP 5.3, we introduced **magic methods**

- Constructors/destructors
- Getters and setters
- Calling methods
- Serialisation hooks
- Etc

Constructors

- `__construct`: called when a new object is instantiated
  - declare any parameters you like
  - usually we inject dependencies
  - perform any other setup

```
class BlueTable {
    public function __construct() {
        $this->colour = "blue";
    }
}
$blue_table = new BlueTable();
echo $blue_table->colour; // blue
```

Destructors

- **`__destruct`**: called when the object is destroyed
    - good time to close resource handles

Fake Properties

When we access a property that doesn't exist, PHP calls **`__get()`** or **`__set()`** for us

```php
class Table {

    public function __get($property) {
        // called if we are reading
        echo "you asked for $property\n";
    }

    public function __set($property, $value) {
        // called if we are writing
        echo "you tried to set $property to $value\n";
    }
}

$table = new Table();

$table->legs = 5;

echo "table has: " . $table->legs . "legs\n";
```

Fake Methods

PHP calls `__call` when we call a method that doesn't exist

```
class Table {
    public function shift($x, $y) {
        // the table moves
        echo "shift table by $x and $y\n";
    }

    public function __call($method, $arguments) {
        // look out for calls to move(), these should be shift()
        if($method == "move") {
            return $this->shift($arguments[0], $arguments[1]);
        }
    }
}

$table = new Table();
$table->shift(3,5); // shift table by 3 and 5
$table->move(4,9); // shift table by 4 and 9
```

There is an equivalent function for static calls, `__callStatic()`

Serialising Objects

We can control what happens when we **`serialize`** and **`unserialize`** objects

```
class Table {
}

$table = new Table();
$table->legs = 4;
$table->colour = "red";

echo serialize($table);
// O:5:"Table":2:{s:4:"legs";i:4;s:6:"colour";s:3:"red";}
```

Serialising Objects

- **`__sleep()`** to specify which properties to store
- **`__wakeup()`** to put in place any additional items on unserialize

```
class Table {
    public function __sleep() {
        return array("legs");
    }
}

$table = new Table();
$table->legs = 7;
$table->colour = "red";

$data =  serialize($table);
echo $data;
// O:5:"Table":1:{s:4:"legs";i:7;}
```

Serialising Objects

- **`__sleep()`** to specify which properties to store
- **`__wakeup()`** to put in place any additional items on unserialize

```
class Table {
    public function __wakeup() {
        $this->colour = "wood";
    }
}

echo $data;
$other_table = unserialize($data);
print_r($other_table);

/* Table Object
(
    [legs] => 7
    [colour] => wood
) */
```

Magic Tricks: clone

Control the behaviour of cloning an object by defining **`__clone()`**

- make it return false to prevent cloning (for a Singleton)
- recreate resources that shouldn’t be shared

Magic Tricks: toString

Control what happens when an object cast to a string. E.g. for an exception

```
class TableException extends Exception {
    public function __toString() {
        return '** ' . $this->getMessage() . ' **';
    }
}

try {
    throw new TableException("it wobbles!");
} catch (TableException $e) {
    echo $e;
}

// output: ** it wobbles! **
```

The default output would be

```
exception ’TableException’ with message ’it wobbles!’  in
/.../tostring.php:7 Stack trace:
```

Design Patterns

Common solutions to common problems. ZCE expects:

- Singleton
- Registry
- Factory
- ActiveRecord
- MVC (Model View Controller)

Singleton

We saw a singleton already

```

class Singleton
{
  private static $classInstance;

  private function __construct () {}

  static function getInstance () {
    if (! isset(self::$classInstance)) {
      self::$classInstance = new Singleton();
    }
    return (self::$classInstance);
  }
}
```

- Only one instance is allowed
- We can’t instantiate it ourselves

Registry

```
class Registry
{
  private static $storage;
    private function __construct () {}
    public function set($key, $value) {
        self::$storage[$key] = $value;
    }
    public function get($key) {
        if(array_key_exists($key, self::$storage)) {
            return self::$storage[$key];
        }
        return false;
    }
}

Registry::set('shinyThing', new StdClass());
// later ...
$shiny = Registry::get('shinyThing');
```

Factory

```
class WidgetFactory
{
  public function getWidget($type) {
    switch($type) {
      case 'DatePicker':
        // assume simple look/feel
        return new SimpleDatePicker(Registry::get('options'));
        break;
      default:
        // do nothing, invalid widget type
        break;
    }
  }
}

$widget_factory = new WidgetFactory();
$picker = $widget_factory->getWidget('DatePicker');
$picker->render();
```

Active Record

- A pattern that hides data access details
- Application simply deals with an object
- Object itself knows how to translate itself to storage

Code can be long/complicated

MVC

- Model-View-Controller
- Separates data access, processing and presentation
- Common in many frameworks today
- Controller retrieves data from models, and passes to appropriate view

Concepts can be tested, code usually isn’t

Classes and Namespaces

- Namespaces help us avoid crazy long classnames
- We can combine libraries with the same classnames
- Our code can be more easily organised

Classes in Namespaces

Declaring the namespace and class:

```

namespace MyLibrary\Logging

class FileLog{
}
```

Using the class from elsewhere (including inside another namespace):

```

$log_handler = new \MyLibrary\Logging\FileLog();
```

Classes in Namespaces

Declaring the namespace and class:

```

namespace MyLibrary\Logging

class FileLog{
}
```

Using the namespace and shortened class name:

```

use \MyLibrary\Logging;
use \MyLibrary\Logging as Fred;

$log_handler = new Logging\FileLog();
$log_handler2 = new Fred\FileLog();
```

Reflection

- An API which allows us to inspect our functions/objects
- Gives meta information
- Includes private/protected properties and methods

Reflecting Functions

```php

function addStars($message) {
    return '** ' . $message . ' **';
}

$reflection = new ReflectionFunction('addStars');

$reflection->getName();
$reflection->getParameters();
$reflection->isUserDefined();
$reflection->getFileName();
```

Reflecting Classes

```php
class Robot {
    public $name;

    public function flashLights($pattern) {
        // look!  Pretty flashing lights
        return true;
    }
}

$reflection = new ReflectionClass('Robot');
$reflection->getMethods();
$reflection->getFileName();
$reflection->getProperties();
$reflection->isInterface());
```

Reflection on the CLI

Reflection gives us these command-line switches:

- **`-rf`** for function information
- **`-rc`** for class information
- **`-re`** for extension information
- **`-ri`** for extension configuration

Not a ZCE question but really useful!

SPL Library

SPL: Standard PHP Library

- Bad news: Huge topic
- Good news: Not much mention in ZCE

SPL: Key Knowledge

- Introduced in PHP 5, new additions in each release
- ArrayObject class
- Standard iterator classes
- Really useful interfaces
  - Countable (we saw earlier)
  - ArrayAccess
  - Iterator
- Data types for storage
- Detailed exceptions
- Autoloading

`http://uk2.php.net/manual/en/book.spl.php`

ArrayAccess

An interface which allows an object to behave like an array

```
abstract public boolean offsetExists ( mixed $offset )
abstract public mixed offsetGet ( mixed $offset )
abstract public void offsetSet ( mixed $offset , mixed $value )
abstract public void offsetUnset ( mixed $offset )
```

Iterator

An interface which defines how an object behaves when "foreach"-ed

```
abstract public mixed current ( void )
abstract public scalar key ( void )
abstract public void next ( void )
abstract public void rewind ( void )
abstract public boolean valid ( void )
```

OOP Resources

- OOP Series: `http://bit.ly/j7yRUa`
- Design Patterns: `http://www.fluffycat.com/PHP-Design-Patterns/`
- MVC: `http://bit.ly/j8Fscu`
- SPL (ramsey): `http://devzone.zend.com/article/2565`
- SPL (Elazar): `http://bit.ly/jiFokK`

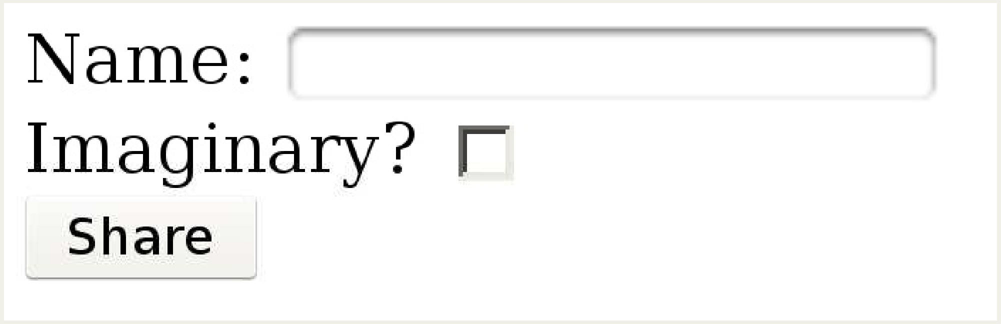
HTTP and the Web

Forms

A form:

```
<form name="myform">
Name: <input type="text" name="item" /><br />
Imaginary?  <input type="checkbox" name="type" value="imaginary" />
<input type="submit" value="Share" />
</form>
```

In the browser:



Submitting Forms

PHP data in **`$_GET`**:

```
Array
(
    [item] => Unicorn
    [type] => imaginary
)
```

If form has **`method="post"`** attribute, data will be in **`$_POST`**

Fun With Forms

- Forms can have many input types:
- For a full list: `http://www.w3schools.com/html/html_forms.asp`

Fun With Forms

- Forms can have many input types:
- For a full list: `http://www.w3schools.com/html/html_forms.asp`
- In the interests of balance: `http://w3fools.com/`

Uploading Files with Forms

- Use **`enctype="multipart/form-data"`** and input type **file**
- Upload information available in **`$_FILES`**
- One element per form file element, containing:
  - name
  - type
  - size
  - tmp_name
  - error
- Config options: **`upload_max_filesize`** and **`upload_tmp_dir`**

HTTP Headers

Headers are request meta-data

Common headers:

- **`Accept`** and **`Content-Type`**
- **`Cookie`** and **`Set-Cookie`**
- **`User-Agent`**
- **`Authorization`**

Headers are sent with both requests and responses

Headers Example

```
curl -I http://www.google.co.uk/
```

```
HTTP/1.1 200 OK
Date: Wed, 04 May 2011 09:50:30 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID=0a902b1fd14bc62f:FF=0:TM=1304502630:LM=1304502630:
Set-Cookie: NID=46=CUminn6rbfPX-oPfF1LQ_PtTpJVvMIeB6q0csmOjv4mnciVY5yP
Server: gws
X-XSS-Protection: 1; mode=block
Transfer-Encoding: chunked
```

Cookies

- Cookies are sent as HTTP headers
- Client returns them on subsequent same-domain requests
- No cookies in first request

```php

// set cookie
setcookie('name', 'Fred', time() + 3600);

// see what cookies we have
var_dump($_COOKIE);
```

Cookie Considerations

- Cookies are invisible to the user
- Data is stored client side
- Easily edited (check your browser options)
- Cannot be trusted

PHP Sessions

Sessions are a better way to store persistent data

- Available by default in PHP
- Start with **`session_start()`** or config **`session.auto_start`**
- Makes a superglobal **`$_SESSION`** available, which persists between requests from the same user

PHP Sessions

Sessions are a better way to store persistent data

- Available by default in PHP
- Start with **`session_start()`** or config **`session.auto_start`**
- Makes a superglobal **`$_SESSION`** available, which persists between requests from the same user
- Session has a unique identifier
- Usually sent to client as a cookie
- Data is stored on the server

Session Storage

- Sessions stored as files in temp directory by default
- Many different handlers available:
  - database
  - memcache
  - ... and extensible
- Set handler with **`session.save_handler`**

Session Functions

- `session_id()`
- `session_regenerate_id()`
- `session_destroy()`

HTTP Authentication

If you’re using Apache, you can use PHP and Basic Authentication

- If credentials were submitted, you’ll find them in
  - **`$_SERVER['PHP_AUTH_USER']`**
  - **`$_SERVER['PHP_AUTH_PW']`**
- To trigger authentication, send a **`WWW-Authenticate: Basic realm=[realm]`** header
- `http://bit.ly/jBeOwb`
- `http://php.net/manual/en/features.http-auth.php`

Web Services and Data Formats

Date and Time

Unix Timestamp: seconds since 1st January 1970

- e.g. 1305656360

Date/Time functions

- **`date()`**
- **`mktime()`**
    - BEWARE arguments **`hour, minute, second, month, day, year`**
- **`strtotime()`**

See: `http://bit.ly/iPyKgv`

DateTime

- OO interface into the same (some better, some fixed) functionality
- Added in 5.2
- Objects
  - DateTime
  - DateTimeZone
  - DateInterval
  - DatePeriod

See: `http://bit.ly/kYuIj9`

XML in PHP

There is (as usual) more than one way to do this

- SimpleXML
  `http://uk2.php.net/manual/en/book.simplexml.php`
- DOM `http://uk2.php.net/manual/en/book.dom.php`

XML in PHP

There is (as usual) more than one way to do this

- SimpleXML
  `http://uk2.php.net/manual/en/book.simplexml.php`
- DOM `http://uk2.php.net/manual/en/book.dom.php`

As a general rule, if SimpleXML can do it, use SimpleXML. Otherwise, use DOM

They are interoperable using **`dom_import_simplexml()`** and **`simplexml_import_dom()`**

SimpleXML

SimpleXML parses XML into a predictable Object structure

- Objects are of type SimpleXMLElement
- Child elements are properties, and themselves are SimpleXMLElement objects
- Where there are multiple same-named children, these become an array*
- Attributes are accessed using array notation
- Does have some limitations (cannot relocate nodes, for example)

* not really, it’s an object with ArrayAccess but it *looks* like an array to us

SimpleXMLElement Functions

Bringing in data:

- **`simplexml_load_file`** - Interprets an XML file into an object
- **`simplexml_load_string`** - Interprets a string of XML into an object

SimpleXMLElement Functions

Manipulating XML

- **`SimpleXMLElement::children`** - Finds children of given node
- **`SimpleXMLElement::attributes`** - Identifies an element's attributes
- **`SimpleXMLElement::addChild`** - Adds a child element to the XML node
- **`SimpleXMLElement::addAttribute`** - Adds an attribute to the SimpleXML element
- **`SimpleXMLElement::getName`** - Gets the name of the XML element
- **`SimpleXMLElement::getDocNamespaces`** - Returns namespaces declared in document
- **`SimpleXMLElement::asXML`** - Return a well-formed XML string based on SimpleXML element

DOM and XML

- More powerful and flexible
- More complex
- Documents represented by **`DOMDocument`** objects

DOMDocument Methods

- **`DOMDocument::load`** - Load XML from a file
- **`DOMDocument::loadXML`** - Load XML from a string
- **`DOMDocument::saveXML`** - Dumps the internal XML tree into a string
- **`DOMDocument::createAttribute`** - Create new attribute
- **`DOMDocument::createElement`** - Create new element node
- **`DOMDocument::getElementsByTagName`** - Searches for all elements with given tag name
- **`DOMDocument::normalizeDocument`** - Normalizes the document

There is also the **`DOMElement`** class

XML Resources

- http://bit.ly/l0EOkz
- http://bit.ly/jPeIKl
- http://devzone.zend.com/article/1713

XPath

Query language for XML, often compared with SQL

- In its simplest form, it searches for a top level tag
- A particular tag inside a tag **`library/shelf`**
- And so on to any level of nesting
- To search for tags at any level in the hierarchy, start with double slash **`//book`**
- To find elements, use an ’at’ sign **`//book@title`**

Both DOM and SimpleXML allow you to perform XPath on child nodes as well as a whole document

JSON

- JavaScript Object Notation
- A string format for representing arrays/objects
- Write it with **`json_encode()`**
- Read it with **`json_decode()`**

JSON Example

```php

$list = array("meat" => array(
    "chicken",
    "lamb",
    "reindeer"),
    "count" => 3);

echo json_encode($list);
```

```
"meat":["chicken","lamb","reindeer"],"count":3
```

Web Services

- Means of exposing functionality or data
- A lot like a web page
- Integration between applications
- Separation within an application
- Works over HTTP, using headers and status codes for additional data
- Can use various data formats, including XML and JSON

RPC Services

These services typically have:

- A single endpoint
- Method names
- Method parameters
- A return value

Soap

- Not an acronym
    - (used to stand for Simple Object Access Protocol)
- Special case of XML-RPC
- VERY easy to do in PHP
- Can be used with a WSDL
    - Web Service Description Language

Publishing a Soap Service

```php

include('Library.php');

$options = array('uri' => 'http://api.local/soap');
$server = new SoapServer(NULL, $options);
$server->setClass('Library');

$server->handle();
```

Consuming a Soap Service

To call PHP directly, we would do:

```php
include('Library.php');

$lib = new Library();
$name = $lib->thinkOfAName();
echo $name; // Arthur Dent
```

Over Soap:

```php
$options = array('uri' => 'http://api.local',
    'location' => 'http://api.local/soap');
$client = new SoapClient(NULL, $options);

$name = $client->thinkOfAName();
echo $name; // Arthur Dent
```

REST

- REST: REpresentational State Transfer
- Can look like "pretty URLs"
- Stateless
- Uses HTTP features
- Can use any data format

In REST, we use HTTP verbs to provide CRUD:

GET	Read
POST	Create
PUT	Update
DELETE	Delete

Using REST

- Every item is a **resource**
- Each resource is represented by a URI
- The "directories" are called **collections**
- We can **GET** items or collections
- To create, we **POST** to the collection
- To update, we **GET** the resource, change it and then **POST** it back to the URI
- To delete, we **DELETE** the resource

Security

Filter Input, Escape Output

Filter input, escape output

Filter Input

- Trust nothing
- Ensure data is type expected
- Whitelist/Blacklist
- **`ctype_*`** functions
- Filter extension

PHP Security Configuration

There are some key ini directives that can help us secure our system

- **`register_globals`**
- **`allow_url_fopen`**
- **`open_basedir`**
- **`disable_functions`**
- **`disable_classes`**

Cross Site Scripting

Someone inserts something malicious into your site that users see, especially if you have user contributed content

Usually javascript, and can be subtle - redirecting users or rewriting links

Filter input, escape output

Input containing scripts should not be accepted, and should **never** be displayed

Cross Site Request Forgery

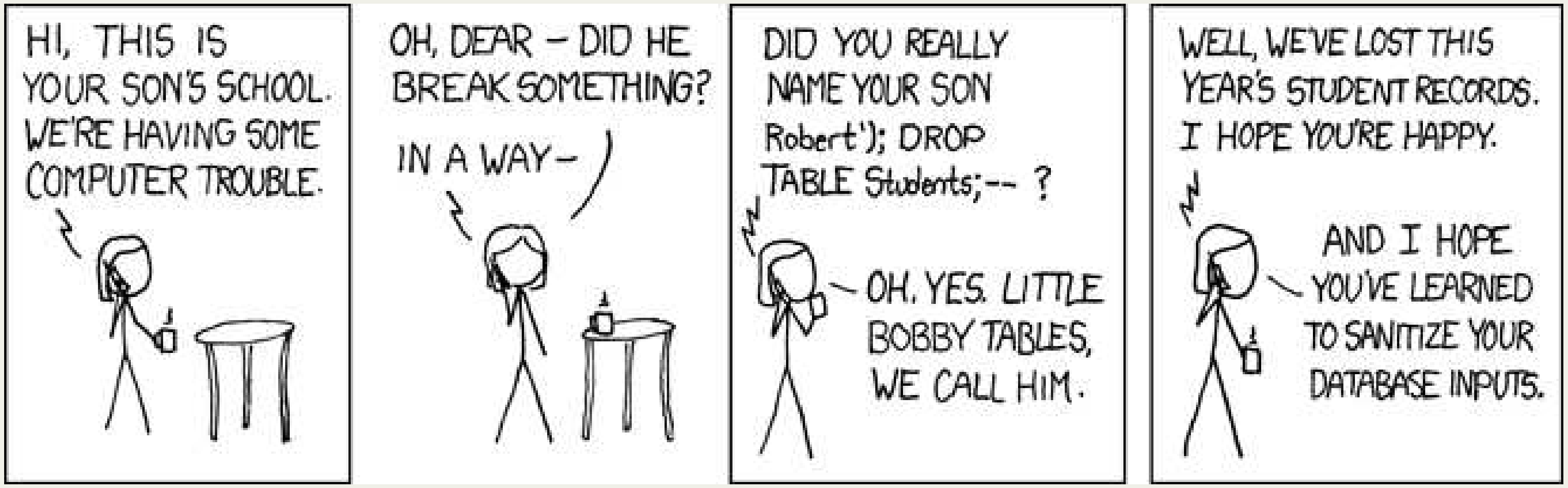
Script makes request to another website

- Uses user privileges
- Invisible to user
- Form submissions that did not come from your forms

To protect:

- Send a unique token with every form
- Only accept a form response if it has the token in it

SQL Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcd.com/327/

SQL Injection

- Again, filter your input!
- SQL injection is passing of unescapted variables to your database
- Use **`*_escape_string()`** to combat it
- PDO and prepared statements protect against it

Databases

SQL

- Assumes MySQL
- DDL queries
- Data manipulation
- PDO

Tables

Creating tables:

```
CREATE TABLE pets (
    pet_id int primary key auto_increment,
    animal varchar(255),
    name varchar(255));
```

Removing tables:

```
DROP TABLE pets;
```

SQL and Data

Inserting data:

```
insert into pets (animal, name) values ("dog", "Rover");
insert into pets (animal, name) values ("cat", "Fluffy");
```

Updating data:

```
update pets set name="Pig" where name="Rover";
```

Deleting data:

```
delete from pets;
```

A where clause can be added too

SQL Joins

A join is when we combine two data sets, e.g. owners and pets

```
+--------+--------+---------+----------+
| pet_id | animal | name    | owner_id |
+--------+--------+---------+----------+
|      1 | dog    | Pig     |        3 |
|      2 | cat    | Fluffy  |        3 |
|      3 | rabbit | blackie |        2 |
|      4 | rabbit | Snowy   |        1 |
|      5 | cat    | Scratch |        3 |
|      6 | cat    | Sniff   |        3 |
+--------+--------+---------+----------+
```

```
+----------+---------+------+
| owner_id | name    | age  |
+----------+---------+------+
|        1 | Jack    |    3 |
|        2 | Jill    |    3 |
|        3 | Harriet |    9 |
+----------+---------+------+
```

SQL Joins: Inner Join

Inner joins join two tables where rows match in both

```
select pets.name, owners.name from pets
inner join owners on pets.owner_id = owners.owner_id;

+---------+---------+
| name    | name    |
+---------+---------+
| Fluffy  | Harriet |
| blackie | Jill    |
| Snowy   | Jack    |
| Scratch | Harriet |
| Sniff   | Harriet |
+---------+---------+
```

A join is an inner join by default

SQL Joins: Left/Right Join

A left join brings all rows from the left column plus matches from the right

```
select pets.name, owners.name from pets
left join owners on pets.owner_id = owners.owner_id;

+---------+---------+
| name    | name    |
+---------+---------+
| Pig     | NULL    |
| Fluffy  | Harriet |
| blackie | Jill    |
| Snowy   | Jack    |
| Scratch | Harriet |
| Sniff   | Harriet |
+---------+---------+
```

A right join is the same but brings all the rows from the right hand side plus any matches on the left

PDO

PDO: PHP Database Objects

- Connects to (many!) various database back-ends
- Replaces the **`mysql_*`** functions and equivalents
- Abstracts database access
- Does not work around SQL differences

`http://uk2.php.net/manual/en/book.pdo.php`

PDO Examples

Fetching data

```
$dbh = new PDO('mysql:host=localhost;dbname=test', 'user', 'pass');

$query = "select name from owners";
$stmt = $dbh->prepare($query);
$success = $stmt->execute();

if($success) {
    while($row = $stmt->fetch()){
        echo "<p>".$row['NAME']."</p>\n";
    }
}
```

Prepared Statements

- Prepared statements standard with PDO
- Use bind variables just as from command line
- These will be sanity checked as they are substituted
- Use placeholders in SQL
- Two types of placeholder
  - :variable
  - ?
- Can also bind to a parameter with **`bindParam()`**

Bind Variables

These are simple placeholders which we substitute values into

```
$dbh = new PDO('mysql:host=localhost;dbname=test', 'user', 'pass');

$sql = 'select * from pets
where animal = ?
and colour = ?';

$stmt = $dbh->prepare($sql);

$stmt->bindValue(1,'cat');
$stmt->bindValue(2,'black');

$stmt->execute();
```

Bind Variables

A more readable but equivalent approach:

```
$dbh = new PDO('mysql:host=localhost;dbname=test', 'user', 'pass');

$sql = 'select * from pets
where animal = :animal
and colour = :colour';

$stmt = $dbh->prepare($sql);

$stmt->bindValue(':colour','rabbit');
$stmt->bindValue(':animal','white');

$stmt->execute();
```

Transactions

- Some database types support transactions
- Transactions are atomic collections of statements
- If all statements complete successfully, transaction is committed
- Otherwise, it is rolled back and none of them ever happened
- PDO supports this
    - **`PDO::beginTransaction()`**
    - **`PDO::commit()`** or **`PDO::rollback()`**

Optimising Queries with EXPLAIN

- Take a query
- Put the **`EXPLAIN`** keyword in front of it
- Gives information about the number of rows scanned to build result set
- Use **`\G`** to make it easier to read

`http://dev.mysql.com/doc/refman/5.0/en/explain.html`

Exam Tips

Timings

- 70 questions approx
- 90 minutes
- 77 seconds on average

Timings

- 70 questions approx
- 90 minutes
- 77 seconds on average

Take your time!

Equipment

You will be allowed to take nothing with you.

They will give you something to write on and with

Scores

- Pass mark is not publicised
- No penalty for a wrong answer
- Some questions worth more marks than others
- You can flag questions to come back to later

Scores

- Pass mark is not publicised
- No penalty for a wrong answer
- Some questions worth more marks than others
- You can flag questions to come back to later

If you don’t know, **GUESS**

Reviewing Questions

When you get to the end of the questions:

- A grid of questions shows
- Unanswered questions are marked
- Flagged questions are marked
- You can go to them, and come back to the grid
- If you haven’t ticked enough boxes, this is shown too

ZCE Benefits

- Right to use "ZCE" and logo
- Entry in Zend Yellow Pages directory
- Software licenses from Zend
- Some employers ask for it
- Bragging rights? :)

And Finally

- Links: `http://bit.ly/ltbYs1`
- Slides: `http://slideshare.net/lornajane`
- Feedback: `http://joind.in/3214`

GOOD LUCK!