**PHP Naming Conventions**

**Names**

**Naming variables, functions, classes and files**

**Variables**

All variables must be lowercase with underscore for "space"

$a = 'something';   
$another\_var = 49;   
$yet\_another\_long\_var = 48;

**Arrays and loops**

Arrays follow the same rule for variables, except data from database should be named $data, and in loops should always have $key => $row pairs

foreach ($data as $key => $row)   
{  
 echo $row['something'];   
}

or if you don't need keys

foreach ($data as $row)   
{  
 echo $row['something'];   
}

**Functions**

Functions must follow camelCaseConvention, which means no spaces and no underscores, lowercase with first letter of each word - uppercase (except the first word)

function fetch ()   
{   
 ...   
}   
  
function anotherFunction ()   
{  
 ...   
}   
  
function yetAnotherLongFunction ()   
{  
 ...   
}

All names should be descriptiove and explain what function does or returns.

**Classess**

Class names are lowercase with each first letter uppercase and underscore ("\_") for space

class Database   
{  
 ...   
}   
  
class Mysql\_Database   
{  
 ...   
}   
  
class My\_Other\_Class\_Name   
{   
 ...   
}

At the end of each class add comment: // end of class   
class User extends Object   
{   
} // end of class

**Files**

Classes should be saved as {$class\_name}.php

require\_once "Database.php";   
require\_once "My\_Other\_Class\_Name.php";

**Common variables**

All counters must be named as *$i*. If you have more than one counter, use the next letter: *$j*, *$k*, *$l*...

for ($i = 0; $i < 100; $i++)

All temporary variables must be called *$temp* or *$tmp;*

for ($i = 0; $i < 100; $i++)   
{   
 $temp = $a[$i];   
 $a[$i] = $b[$i];   
 $b[$i] = $temp;   
}

**Function names**

Try to use as descriptive names as possible, but avoid to many words or to long names.

If you have class User and function to validate if this use is valid or not, instead of this

if (User::isValidUser ())

Or even worse

if (User::checkIfThisUserIsValidAndActivated ())

Use this

if (User::isValid ())

Becauses there's no need to reapeat word "User" since it's already in the class name

**Function prefix**

All functions that return some $variable should be prefixed with "**get**".

$username = getUsername (); $  
data = $object->getLastItem ();   
$user = Class\_Name::getUserById ($id);

All functions that set object variable should be prefixed with "**set**"

setUsername ($username);   
$object->setLastItem ($item);   
Class\_Name::setUserId ($id);

**Constants**

**Global Constants**

All global constants (constants that are used everywhere in application, not just single class) must be uppercase

define ('USER\_TIMEOUT', '3000');   
define ('LIMIT\_POSTS', 15);

**Class constants**

Constants that are used in a single class should be defined as class constants, not global constants. Uppercase but without prefix, and must be defined first before any other class variable

class User   
{   
 const LIMIT = 2;   
 const ANOTHER = 'test';   
}   
echo User::LIMIT;

**Test Values**

**Dummy variables**

All test variables should follow those rules

$url = "www.example.com";   
$string = "dummy"; // dummy strings   
$temp = 'some temp value'; // temporary values

# PHP Coding Standards and Naming Conventions

### Basic Requirements

PHP 5 code is required.

All code must be E\_STRICT-compatible. This means that it must not produce any warnings or errors when PHP's error reporting level is set to E\_ALL | E\_STRICT.

### Layout and indentation

Indenting code is a must. Tabs size is 4 characters and your editor must insert TAB as TAB not as 4 SPACES

#### Array keys

Array keys must be quoted with single quote (')

echo $row['something'];

#### Strings

Strings can be with single or double quote

$s = "This is 1st example";   
$s = 'This is 1st example';

#### PHP variables in string

Must be in brackets, instead of this:

$s = "This is $variable example";

Use this

$s = "This is {$variable} example";

or

$s = 'This is '.$variable.' example';

#### Comments

Comment code ad much as possbile and as long as it makes sense.

/\*\*   
\* This is what function does   
\*   
\* @access public   
\* @param $var1 (string) - username   
\* @param $var2 (int) - user id   
\* @return bool   
\*/   
public function getUser ($var1, $var2 = 0)   
{   
 // do something   
 return true;   
}

Also use one line comments when needed

$activated = User::isValid(); // check if user activated account

No need to comment obvious things

while ($i < $something)   
{   
 ...   
 $i++; // increment $i by one <- **THIS IS STUPID COMMENT**   
}

### Basic rules

Equal sign (=) must have at least one space before and one after

$a = 'something';

If you have more assignments one after another, try to align =

$a = 'something';   
$another = 45;

Compare sign (==, =>, >, <, <=, !=) also need minimum one space before and after

if ($condition == 'I know')

#### If else

Always use brackets { } even for simple if () statement

if ($condition == CONSTANT)   
{   
 $this->doSomethingCool ($condition);   
}

If you have more similar if/else statements, you can align brackets but you still have to use brackets:

if ($condition == CONSTANT)   
{   
 $this->doSomethingCool ($condition);   
}   
elseif ($condition = 24)   
{   
 Object::staticFunction ();   
}   
else   
{   
 $this->doOtherThing ();   
}

Between "if", "for", "switch", "foreach" and () must be one space

###### Example

if ($condition == CONSTANT)   
{   
 $this->doSomethingCool ($condition);   
}   
elseif ($condition = 24 || !empty($stuff))   
{   
 Object::staticFunction ();   
}   
elseif ($condition == 'I know')   
{   
 Another\_Object::newStaticFunction ();   
}   
else   
{  
 $this->doOtherThing ();   
}

public static function functionName ($variable\_one, $variable\_two = '')   
{   
 if ($something)   
 {  
 $this->get();   
 }

}

##### Multiple conditions

If you have if statements with long multiple conditions try to separate in different lines

if ($something == 'Something' && $another\_thing > 45)   
{  
 // do something   
}

For more complex conditions split in 2 lines

if (($something == 'Something' && $another\_thing > 45) || (User::isValid() && $product->id))   
{  
 // do something   
}

Or even better

if ( ($something == 'Something' && $another\_thing > 45) || (User::isValid() && $product->id) )   
{  
 // do something   
}

#### Conditions order

If you have multiple conditions put the most common condition first.

if (!empty($var) && $var == 'something')

So the second condition never checks if the first one is false ( $var is empty )

#### Switch

switch ($condition)   
{  
 case 'something':   
 // do something here   
 break;   
  
 case 48:   
 // do the stuff   
 // no break; - if you need case with no "break" you have to write   
comment   
  
 case SOME\_CONSTANT:   
 default:   
 // do some magic   
 break;   
}

### Functions

#### Function variables

All function variables myst be

* Required variables first
* Varibables with default values last

function someFunction (**$var1**)   
{  
 // function body   
}

function otherFunction (**$var1, $variable\_two**)   
{  
 // function body   
}

function thirdFunction (**$var1, $variable\_two = 'default'**)   
{  
 // function body   
}

Functions with default variables as first parameters are just plain stupid

function thirdFunction (**$var1 = 'default'**, $variable\_two)   
{  
 // function body   
}

Try to avoid more than 4-5 function parameters, but if you absolutely need it, then separate in several lines:

function thirdFunction ($var1, $another\_variable, $third\_variable, $yet\_another\_variable = 44, $variable\_two = 'default')   
{   
 // function body   
}

### Loops

#### Array loops

For single dimension arrays user $val for array values and $key for array keys

foreach ($item as $key => **$val**)   
{   
 // do something here   
}

#### Multidimensional array loops

For multi dimension arrays user $row for array values and $key for array keys

foreach ($data as $key => **$row**)   
{   
 // do something here   
}

#### Nested loops

foreach ($data as $key => $row)   
{   
 // do something here   
 foreach ($row as $k => $v)   
 {   
 // do this for this row   
 }   
}

#### For loops and counters

for ($i = 0; $i < 5; $i++)   
{  
 // do something   
}

#### Nested For loops

for ($i = 0; $i < 5; $i++)   
{  
 // do something   
 for ($j = $i; $j < 100; $j++)   
 {   
 // do something   
 }   
}

### Operators

Always use space between operators

$a=$something\*$b+$i

Replace it with:

$a = $something \* $b + $i;

Except for string concatenation

$s = 'This is my '.$order.' example';

Or

$s = 'And this is my '.($order + 1).' example';

Or

$s = 'And this is my '.round (max ($order / 100) \* 100).'% example';

### Exceptions

try {   
 if (!App::dosomething())   
 {   
 throw new Exception('Hey, we screwed up');   
 }   
 // do whatever you wanted to do   
} catch (Exception $e) { customExceptionHandler ($e); }

### Include() and require()

Include and require are statements not functions therefore use it like this:

include "/path/to/file/some.php";   
require "/path/to/file/some.php";

Prefered methods are require\_once and include\_once

include\_once "/path/to/file/some.php";   
require\_once "/path/to/file/some.php";

# MySQL Coding Standards and Naming Conventions

General rules for MySQL database tables, fields, indices and queries.

### MySQL Naming Convention

#### Engine and charset

Engine is always MyISAM except in special case when required different. All databases must use UTF8 charset.

#### Database

Database name must have the same name as project name. If the project name is "My Web Site" database should be created as:

CREATE DATABASE mywebsite DEFAULT CHARSET UTF8;

#### Tables

All tables must be UTF8 encoded. All tables in the same package must have prefix. Use 2 or 3 letters that describe the package to prefix database. If the project name is Real Deal Marketing, the most obvious prefix will be "rdm\_";

CREATE TABLE rdm\_affiliates;

#### ALTER TABLE and file versions major.minor[.build[.revision]]

All alterations for all tables should be saved in a (database\_name).sql file. If database name is "realdealmarketing" filename should be realdealmarketing-1.0.0.sql

1. Initial file must be version 1.0.0
2. If there are only **table alterations** file should have new version increased by 0.0.1 Example: realdealmarketing-1.0.7.sql
3. If **new tables are created**, version must be increased by 0.1. Example: realdealmarketing-1.2.0.sql
4. If **tables are dropped**, version must be increased by 1.0. Example: realdealmarketing-2.0.0.sql
5. All **minor revisions** should have appropriate file name and version following this pattern: major.minor[.build[.revision]]
6. All versions should be saved in the same directory

#### Fields

This section explains how to create database table fields and how to choose name for each field.

##### Field Names

1. Field names must be prefixed with 2-4 letters of table name.
2. Field names are always lowercase with "\_" to separate words

###### Example

DROP TABLE IF EXISTS rdm\_affiliates;   
CREATE TABLE IF NOT EXISTS rdm\_affiliates   
(   
  **aff\_**id INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,   
  **aff\_**url VARCHAR(120) NOT NULL DEFAULT '',   
  **aff\_**title VARCHAR(120) NOT NULL DEFAULT '',   
  **aff\_**website CHAR(10) NOT NULL DEFAULT '',   
  **aff\_**gender CHAR(8) NOT NULL DEFAULT '',   
  **aff\_**landing\_page VARCHAR(120) NOT NULL DEFAULT '',   
  **aff\_**link VARCHAR(120) NOT NULL DEFAULT '',   
  **aff\_**text TEXT,   
  
 PRIMARY KEY (aff\_id),   
 INDEX (aff\_website)   
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

All field names must be descriptive, avoid names like "unique", "sort" and reserved words.

##### NULL and DEFAULT

All fields except AUTO\_INCREAMENT, TEXT, DATE (and similar) must be defined as **NOT NULL DEFAULT** 'value'

Examples:

aff\_title VARCHAR(120) **NOT NULL DEFAULT ''**   
aff\_gender ENUM('male','female') **NOT NULL DEFAULT 'male'**   
user\_id INT UNSIGNED **NOT NULL DEFAULT 0**   
user\_birthday DATE DEFAULT NULL   
user\_allow\_newsletter TINYINT(1) UNSIGNED **NOT NULL DEFAULT 0**   
art\_text TEXT

#### Indices

All fields in WHERE and ORDER BY should be defined as INDEX. If you have query like this:

SELECT \* FROM table1 WHERE a='something' ORDER BY b

Then fields a, b should be considered as indices

INDEX index\_name (a,b)

##### CHAR Indices

If text field (CHAR, VARCHAR) is used as index for larger tables (> 5000 records) should be always defined as CHAR. For example, if you have fields like this:

aff\_url **VARCHAR(255)** NOT NULL DEFAULT ''   
...   
INDEX afiliate\_url (aff\_url)

To improve performance, this table should be changed to something like this

aff\_url **CHAR(20)** NOT NULL DEFAULT ''   
...   
INDEX afiliate\_url (aff\_url)

#### Queries

Reserved words should be uppercase to increase readability. Also try to separate long queries in multiple lines, while simple queries should stay in one line. Example:

**SELECT** a.field\_name1, a.field\_name2, **COUNT**(a.field3) AS cnt, b.\*   
 **FROM** table1 **AS** a   
 **LEFT JOIN** table2 **AS** b **ON** (a.key\_field = b.key\_field)   
 **RIGHT JOIN** table3 **AS** c **ON** (a.key\_field2 = c.key\_field2)   
 **WHERE** a.field7 = 'something here' **AND** b.field9 = '45'   
 **GROUP BY** a.field\_name1   
 **ORDER BY** a.field\_name2 **DESC**, b.field8   
 **HAVING** cnt > 5

##### Escape

Always escape values in WHERE even when value is integer

##### PHP variables in queries

PHP variables must be defined and escaped before query.

###### Avoid:

$q = "SELECT \* FROM users WHERE email='**".\_escape($\_POST['email'])."**' ";

###### Use:

$email = \_escape($\_POST['email']);   
if (empty($email))   
{   
 return false;   
}   
$q = "SELECT \* FROM users WHERE email='**{$email}**' ";

## Appendix

### PHPMyAdmin

Try to avoid PHP My Admin for table creation because it has problems with default values. It's great product for browsing and simple database manipulation but it makes you lazy and you usually forget to keep history of table alteration and other changes.