Code Style Statement

**Введение**

Этот документ описывает единый стиль кода PHP.

**Общие правила**

Для соединения нескольких слов мы используем camelCase. В именах переменных, функций, классов, методов и т. д. не должно быть никакого транслита.

**Пустые строки**

Пустыми строками мы отделяем циклы, функции, блоки условных операторов, по пустой строке сверху и снизу.

Пример:

if(...){…}

while(...){...}

switch(...){…}

**Пробелы в строке**

Пробелы в строке используются для отделения операторов.

Пример:

$taxOfYear = $tax \* 12;

**Комментарии**

Для описания сути некоторого участка кода, пояснений к алгоритму и другой важной информации используйте несколько подряд идущих однострочных комментариев (//...). Между группой комментариев и собственно кодом поставьте пустую строку. Это покажет, что комментарий относится к блоку кода, а не к конкретной инструкции. Напротив, если комментарий относится к конкретной инструкции, прижмите его вплотную к этой инструкции.

Отделяйте текст комментария одним пробелом «// Текст комментария.».

Комментируя код, старайтесь объяснять, что он делает, а не какая операция производится. Так, инструкции if соответствует выражение «если... то...», причем часть, идущая за «то», является кодом, который будет выполняться, если выражение в if будет верным. Таким образом, для конструкции «if (somePath && File.Exists(somePath))», нужно написать комментарий «// Если выбранный файл существует, то...», а не «// Производим проверку на наличие файла и, если он имеется, удаляем его». Часть предложения, идущую за «то», вписывайте непосредственно перед выполнением конкретных действий. Для инструкций, осуществляющих действия, пишите «// Производим...» или «// Делаем...», где вместо троеточия вписывайте описания действий. Описывая действия, старайтесь описывать суть происходящего, а не то, что делают те или иные операторы. Так, совершенно бессмысленны комментарии вроде «Присваиваем переменной a значение b» или «вызываем метод f».

Помните, что экономить на комментариях нельзя. Однако не стоит также формально подходить к процессу создания комментариев. Задача комментария – упростить понимание кода. Есть немало случаев, когда сам код отличным образом себя документирует.

**Именование переменных**

Имя переменной должно максимально точно отображать цель ее создания. Называя переменную, старайтесь давать ей не слишком обобщенное имя. Идите в сторону конкретизации. Нужно четко представлять, для чего она предназначена. Даже если вам при обобщении это понятно, далеко не факт, что это будет очевидно для других разработчиков в вашей команде.

При составлении имени переменной необходимо «зашивать» тип данных, хранимый в этой переменной. Тип переменной указывается в конце имени через \_.

integer - int

float - fl

string - str

boolean - bl

array - arr

object - obj

resourse - res

За все время использования переменная не должна менять свой тип. При необходимости можно использовать явное приведение во время операции:

$userPayment\_int = (int)$taxOfYear\_fl \* 2;

Если надо сменить тип переменной: создаем новую переменную с другим типом, старую удаляем.

Пример:

$priceOfGood\_int = 4300;

$priceOfGood\_fl = (float)$priceOfGood\_int;

unset($priceOfGood\_int);

Несмотря на громоздкость такого принципа, позволяет избежать трудноловимых багов, к тому же подобная задача встречается редко.

***Пример хороших имен***

$employeesCount\_int, $userMonthWorkDaysCount\_int, $yearTax\_fl, $maxComputedSalary\_int

***Пример плохих имен***

x, $fb, f\_ar\_type\_data, f\_obj\_result, inputData, returnValue, resultArray, obj1, obj2

Для переменных-счетчиков коротких циклов допускаются традиционные односимвольные имена — i, j, k, …. Если переменная будет использоваться вне цикла, ее имя должно быть более выразительным и понятным:

$teamScoresCount = 0;

$teamScores = [];

Избегайте именования переменных ничего не значащими (значащими только для вас) именами (snuffi, elizabeth) если только они не являются представляемыми в программе сущностями.

Имена булевых переменных должны быть в утвердительной форме и подразумевать булево значение: done, error, found, finished. Плохо: status, source.

Использовать каждую переменную только с одной целью. Не стоит создавать переменную, которая вначале кода хранит имя пользователя, а в конце его фамилию.

**Инструкции**

**Условные операторы**

**if**

Пример:

if(...){

//code

}

**if..else**

Пример:

if(...){

//code

}else{

//code

}

**if..elseif**

if(...){

//code

}elseif(...){

//code

}else{

//code

}

**Циклы**

Это нормально, когда небольшой цикл из 1-3 строк имеет индекс под названием I,j или k. Но если тело цикла заметно больше, то лучше давать индексам осмысленные имена. И вам будет проще разобраться с таким кодом со временем (сразу же становится понятно, что делает цикл) и другим программистам, которые будут работать с вашим кодом, тоже станет легче.

for($i=0; $i<7; $i++){

//code

}

**switch**

switch (...) {

case 'value':

// code...

break;

default:

// code...

break;

}

**try...catch**

try {

//code

} catch (Exception $e) {

//code

}

**Функции**

В название функции надо «зашить» тип возвращаемых данных по тем же правилам что и для переменной. Пример оформления функции ниже:

function showNewUser\_arr(){

//code

}

Если функция возвращает более 1-го типа данных «зашиваем» тип данных — mix

Пример:

function showNewUser\_mix(){

//code

}

Если функция ничего не возвращает — используем void

function showNewUser\_void(){

//code

}

**Классы**

Классы всегда именуем с большой буквы. Назначение класса и краткое описание методов должно быть написано в комментарии, непосредственно перед объявлением класса. Назначение методов класса описывается тезисно(кратко). Сначала перечисляем публичные методы, потом защищенные, потом приватные. Свойства объявляем внутри класса и подписываем комментарием, порядок объявления как для методов — public, protected, private. При использовании трейтов, тезисно описываем каждый. Пример:

/\*\*

\* Класс администратора

\* generatePasswordHash - генерирует хеш пароля

\* getAdminId - получает id админа

\*/

class Admin extends User

{

use Auth;

// Auth - авторизация

public $adminName; // имя админа

protected $adminId; // id админа

public function getAdminId\_int(){...}

protected function generatePasswordHash\_str($passwd){...}

}

#### Качественные методы

##### Метод должен служить одной четко определенной цели

Эта цель должна быть полностью отражена в его имени: получить текущего пользователя — **getСurrentUser\_str()**. Размытое, неоднозначное и откровенно плохое имя метода чаще всего является главным свидетельством его неудачной реализации.  
  
**Примеры хороших имен методов:**

Customer::getFullName\_str() – получить полное имя клиента UserMapper::createAndGetUser\_str(userId) – исключение, в контексте User-маппера побочная роль метода (вернуть вновь созданный user-объект) достаточно очевидна.

**Примеры плохих имен методов:**

computeMonthRevenueAndDoExport() – несколько несвязанных целей  
processInput(), handleCalculation() – невыразительность, размытость цели  
метода

* Следует использовать парные антонимы для методов, выполняющих парные (противоположные) действия: open/close, show/hide, add/remove, insert/delete.
* Метод должен быть отформатирован в строгом соответствии с принятыми конвенциями, «код стайлами» и так далее.
* Метод должен быть документирован комментарии к важным и сложным участкам).
* Метод не должен изменять глобальные переменные.
* Метод не должен иметь слишком большое количество параметров (не более 7).
* Параметры следует упорядочивать по степени их важности либо порядку их использования внутри метода:

setMonthExchangeRate\_void($month\_int, $exchangeRate\_fl)  
getCustomerMonthRevenue\_int($customerId\_str, $month\_int)

* Не должно быть неиспользуемых «брошенных» переменных и параметров.
* Метод должен быть защищен от плохих данных, которые могут нарушить его работу.

$monthRevenue\_fl = $fixedValue\_int \* 0.6 / $inputParam\_fl

* Метод не должен быть слишком большим (не более 100 строк).

**SQL запрос**

Разбиваем на блоки:

'SELECT

\*

FROM

tbl

WHERE

a = 1';