Шаблонизаторы.

Оглавление

[Основы 1](#_Toc462665141)

[Приступаем к делу 2](#_Toc462665142)

[Условные операторы в Twig 3](#_Toc462665143)

[Циклы в Twig 6](#_Toc462665144)

[Дамп данных 9](#_Toc462665145)

[Подгрузка шаблонов 12](#_Toc462665146)

[Фильтрация данных 14](#_Toc462665147)

[Исключения (Exceptions) 17](#_Toc462665148)

[Домашнее задание 19](#_Toc462665149)

[Дополнительные материалы и литература 19](#_Toc462665150)

На курсе PHP Level 1 применялся шаблонизатор, который мы написали самостоятельно. Он выполнял базовый функционал подстановки значений в заранее подготовленные блоки шаблонов и мог работать с простыми циклами. Однако зачастую требуется более мощный функционал при меньших трудозатратах по подготовке. Поэтому мы рассмотрим готовый шаблонизатор, который при желании легки применим в нашем движке интернет-магазина.

Как правило, все шаблонизаторы похожи друг на друга и различаются только какими-то специфическими особенностями и деталями. Главная задача шаблонизаторов - разделить бизнес логику приложения и вывод данных на страницу, таким образом, позволив разработчикам и дизайнерам работать одновременно.

Многие PHP фрэймворки, включая: Zend Frameworkd, Symfony, Yii2 и другие по-своему реализуют разделение бизнес логики и вывод данных. Однако, если вы не любите фреймворки или ваш проект слишком мал для их использования, то вы можете воспользоваться какой-то отдельной системой построения шаблонов. Мы будем рассматривать шаблонизатор Twig.

**Установка Twig**

Существует множество способов установки Twig-a. Самый простой и быстрый - это скачивание компонента с [GitHub](https://github.com/fabpot/Twig/archive/master.zip), после чего архив необходимо распаковать, и перекинуть каталог lib в папку с нашим проектом.

# Основы

Прежде чем приступить непосредственно к делу, давайте разберёмся с принципом работы шаблонизаторов. Обычное PHP приложение состоит из целого набора страниц, которые включают в себя как статический HTML код (меню, списки, изображения и т.д.), так и динамический контент (вывод данных из БД, xml файла, сервисы, …). С помощью Twig мы можем разделить данные процессы, создавая шаблоны со специальными маркерами, вместо которых в последствии будет вставляться динамический контент.

Значения для данных маркеров формируются в основном PHP скрипте; там же происходит общение с базой данных, xml парсинг и другие всевозможные операции. Таким образом, ваша страница будет строиться на основе 2х источников: шаблона с специальными вставками и PHP скриптов, где мы храним основной функционал. Это даёт возможность PHP разработчикам и дизайнерам одновременно работать над одними и теми же страницами.

# Приступаем к делу

Для того чтобы посмотреть, как работает Twig, предлагаю рассмотреть простой пример:

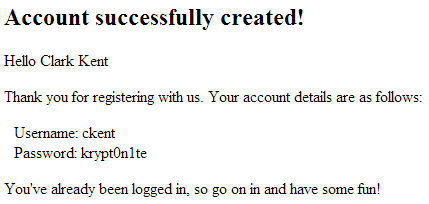
|  |
| --- |
| <html>  <head></head>  <body>  <h2>Account successfully created!</h2>  <p>Hello {{ name }}</p>  <p>Thank you for registering with us. Your account details are as follows: </p>  <p style="margin-left: 10px">  Username: {{ username }} <br/>  Password: {{ password }}  </p>    <p>You've already been logged in, so go on in and have some fun!</p>  </body>  </html> |

Обратите внимание на то, что все маркеры, представляющие собой переменные, помещены в двойные фигурные скобки. Подобная запись подскажет Twig-у, где и как осуществлять вставку данных. В предудщей версии нашего движка использовался такой же синтаксис.

Затем, нам необходимо создать основной скрипт, где будет происходить формирование переменных и данных:

|  |
| --- |
| <?php  // подгружаем и активируем авто-загрузчик Twig-а  require\_once 'Twig/Autoloader.php';  Twig\_Autoloader::register();    try {  // указывае где хранятся шаблоны  $loader = new Twig\_Loader\_Filesystem('templates');    // инициализируем Twig  $twig = new Twig\_Environment($loader);    // подгружаем шаблон  $template = $twig->loadTemplate('thanks.tmpl');    // передаём в шаблон переменные и значения  // выводим сформированное содержание  echo $template->render(array(  'name' => 'Clark Kent',  'username' => 'ckent',  'password' => 'krypt0n1te',  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |

В результате, если вы откроете данную страницу в браузере, то увидите следующее:



Таким образом, для использования Twig-а, вам нужно пройти следующие шаги:

1. Инициализировать авто-загрузчик Twig-а, для того чтобы классы шаблонизатора подгружались автоматически.
2. Инициализировать загрузчик шаблонов. В нашем случае эт Twig\_Loader\_FileSystem. В качестве аргумента передаём путь к каталогу с шаблонами.
3. Создать объект самого Twig и передать ему уже сконфигурированные настройки.
4. Подгрузить нужный нам шаблон с помощью метода loadTemplate, передав в него название используемого шаблона. В качестве результата метод вернёт экземпляр шаблона.
5. Сформировать массив вида "ключ-значение", где ключи - это названия переменных, а значения - данные, выводимые в шаблоне. Затем этот массив нужно передать в метод render(), который совместит шаблон с переданными данными и вернёт сгенерированный результат.

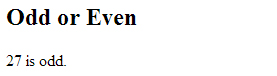
Условные операторы в Twig

Twig также предоставляет нам возможность создавать условные выражения ‘if-else-endif’. Пример:

|  |
| --- |
| <html>  <head></head>  <body>  <h2>Odd or Even</h2>  {% if div == 0 %}  {{ num }} is even.  {% else %}  {{ num }} is odd.  {% endif % }  </body>  </html> |

В зависимости от числа, которое генерируется в основном PHP скрипте, шаблон отобразит одно из двух сообщений. Вот и скрипт, где генерируется число от 0 до 30 и проверяется на чётность

|  |
| --- |
| <?php  include 'Twig/Autoloader.php';  Twig\_Autoloader::register();    try {  $loader = new Twig\_Loader\_Filesystem('templates');    $twig = new Twig\_Environment($loader);    $template = $twig->loadTemplate('numbers.tmpl');    // генерируем случайное число  // и проверяем его на чётность  $num = rand (0,30);  $div = ($num % 2);    echo $template->render(array (  'num' => $num,  'div' => $div  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |

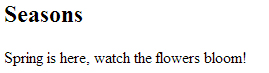


Также мы можем сделать многоуровневые проверки ‘if-elseif-else-endif’. Пример:

|  |
| --- |
| <html>  <head></head>  <body>  <h2>Seasons</h2>  {% if month > 0 and month <= 3 %}  Spring is here, watch the flowers bloom!  {% elseif month > 3 and month <= 6 %}  Summer is here, time to hit the beach!  {% elseif month > 6 and month <= 9 %}  Autumn is here, watch the leaves slowly fall!  {% elseif month > 9 and month <= 12 %}  Winter is here, time to hit the slopes!  {% endif %}  </body>  </html> |

А вот и скрипт, где мы генерируем номер месяца и передаём его в шаблон:

|  |
| --- |
| <?php  include 'Twig/Autoloader.php';  Twig\_Autoloader::register();    try {  $loader = new Twig\_Loader\_Filesystem('templates');    $twig = new Twig\_Environment($loader);    $template = $twig->loadTemplate('seasons.tmpl');    // получаем номер месяца  $month = date('m', mktime());    echo $template->render(array (  'month' => $month  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |



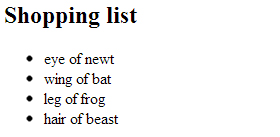
Циклы в Twig

Twig также поддерживает цикл ‘for’. Как Вы уже знаете, он очень удобен, если нам необходимо пройтись по массиву. Пример:

|  |
| --- |
| <html>  <head></head>  <body>  <h2>Shopping list</h2>  <ul>  {% for item in items %}  <li>{{ item }}</li>  {% endfor %}  </ul>  </body>  </html> |

В данном примере у нас простой не ассоциативный массив. На каждой итерации мы будем получать по одному элементу и выводить его в элементе списка. Вот и скрипт:

|  |
| --- |
| <?php  // формируем массив  $items = array(  'eye of newt',  'wing of bat',  'leg of frog',  'hair of beast'  );    include 'Twig/Autoloader.php';  Twig\_Autoloader::register();    try {  $loader = new Twig\_Loader\_Filesystem('templates');    $twig = new Twig\_Environment($loader);    $template = $twig->loadTemplate('list.tmpl');    echo $template->render(array (  'items' => $items  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |

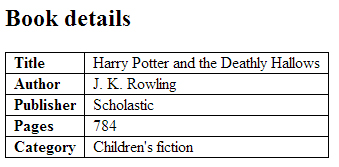


Для того чтобы пройтись по ассоциативному массиву, мы можем обращаться к ключам через “точку”. Пример: пройтись по массиву. Пример:

|  |
| --- |
| <?php  // готовим ассоциативный массив  $book = array(  'title' => 'Harry Potter and the Deathly Hallows',  'author' => 'J. K. Rowling',  'publisher' => 'Scholastic',  'category' => 'Children\'s fiction',  'pages' => '784'  );    include 'Twig/Autoloader.php';  Twig\_Autoloader::register();    try {  $loader = new Twig\_Loader\_Filesystem('templates');    $twig = new Twig\_Environment($loader);    $template = $twig->loadTemplate('book.tmpl');    echo $template->render(array (  'book' => $book  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |

Для того чтобы достучаться до значений массива в шаблоне, сначала пишем имя переменной, в которой хранится сам массив. Затем ставим точку и пишем название ключа, по которому достаём данные:

|  |
| --- |
| <html>  <head>  <style type="text/css">  table {  border-collapse: collapse;  }  tr.heading {  font-weight: bolder;  }  td {  border: 1px solid black;  padding: 0 0.5em;  }  </style>  </head>  <body>  <h2>Book details</h2>  <table>  <tr>  <td><strong>Title</strong></td>  <td>{{ book.title }}</td>  </tr>  <tr>  <td><strong>Author</strong></td>  <td>{{ book.author }}</td>  </tr>  <tr>  <td><strong>Publisher</strong></td>  <td>{{ book.publisher }}</td>  </tr>  <tr>  <td><strong>Pages</strong></td>  <td>{{ book.pages }}</td>  </tr>  <tr>  <td><strong>Category</strong></td>  <td>{{ book.category }}</td>  </tr>  </table>  </body>  </html> |



Такой же подход может быть применён для работы с объектами.

Дамп данных

Безусловно циклы вам пригодиться при выводе данных из БД. Пример:

|  |
| --- |
| <html>  <head>  <style type="text/css">  table {  border-collapse: collapse;  }  tr.heading {  font-weight: bolder;  }  td {  border: 1px solid black;  padding: 0 0.5em;  }  </style>  </head>  <body>  <h2>Countries and capitals</h2>  <table>  <tr class="heading">  <td>Country</td>  <td>Region</td>  <td>Population</td>  <td>Capital</td>  <td>Language</td>  </tr>  {% for d in data %}  <tr>  <td>{{ d.name|escape }}</td>  <td>{{ d.region|escape }}</td>  <td>{{ d.population|escape }}</td>  <td>{{ d.capital|escape }}</td>  <td>{{ d.language|escape }}</td>  </tr>  {% endfor %}  </table>  </body>  </html> |

В следующем фрагменте кода я использую PDO подключение к MySQL базе данных ‘world’. Если вы хотите попробовать данный пример, то вам нужно сформировать базу самим:

|  |
| --- |
| <?php  include 'Twig/Autoloader.php';  Twig\_Autoloader::register();    // подключение к бд  try {  $dbh = new PDO('mysql:dbname=world;host=localhost', 'root', 'guessme');  } catch (PDOException $e) {  echo "Error: Could not connect. " . $e->getMessage();  }  // установка error режима  $dbh->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);    // выполняем запрос  try {  // формируем SELECT запрос  // в результате каждая строка таблицы будет объектом  $sql = "SELECT country.Code AS code, country.Name AS name, country.Region AS region, country.Population AS population, countrylanguage.Language AS language, ity.Name AS capital FROM country, city, countrylanguage WHERE country.Code = city.CountryCode AND country.Capital = city.ID AND country.Code = countrylanguage.CountryCode AND countrylanguage.IsOfficial = 'T' ORDER BY population DESC LIMIT 0,20";  $sth = $dbh->query($sql);  while ($row = $sth->fetchObject()) {  $data[] = $row;  }    // закрываем соединение  unset($dbh);    $loader = new Twig\_Loader\_Filesystem('templates');    $twig = new Twig\_Environment($loader);    $template = $twig->loadTemplate('countries2.tmpl');    echo $template->render(array (  'data' => $data  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |

Тут стоит отметить несколько вещей:

Мы используем метод fetchObject(), который вернёт нам строки из таблицы в виде объектов. Названия полей будут соответствовать названиям колонок. Затем эти объекты мы помещаем в массив и передаём его в шаблон. В шаблоне используем цикл и выводим данные.

В данном примере также используется встроенный в Twig фильтр `escape`. По умолчанию данный фильтр пользуется функцией htmlspecialchars() для фильтровки данных. Это неплохая защита от XSS атак.

Подгрузка шаблонов

Также в Twig-е есть ещё одна команда - `include`, которая позволяет подключать содержание других шаблонов. Это может пригодиться, когда вы захотите прикрепить к вашим файлам меню, заголовок или подвал.

Для демонстрации представьте, что данный код - это главный шаблон:

|  |
| --- |
| <html>  <head>  <link rel="stylesheet" type="text/css" href="main.css" />  </head>  <body>  <div id="page">  <div id="header">  {% include 'primary.tmpl' %}  </div>    <div id="left">  {% include 'secondary.tmpl' %}  </div>    <div id="right">  This is the main page content. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.  </div>    <div id="footer">  {% include 'footer.tmpl' %}  </div>  </div>  </body>  </html> |

Все секции данной страницы находятся в отдельных файлах и подключаются сюда с помощью команды `include`. Давайте посмотрим, как выглядят подключаемые файлы:

|  |
| --- |
| <!-- begin: primary.tmpl -->  <table>  <tr>  {% for item in nav.primary %}  <td><a href="{{ item.url }}">{{ item.name|upper }}</a></td>  {% endfor %}  </tr>  </table>  <!-- end: primary.tmpl --> |
| <!-- begin: secondary.tmpl -->  <ul>  {% for item in nav.secondary %}  <li><a href="{{ item.url }}">{{ item.name }}</a></li>  {% endfor %}  </ul>  <!-- end: secondary.tmpl --> |
| <!-- begin: footer.tmpl -->  <div style="align:center">  This page licensed under a Creative Commons License. Last updated on: {{ updated }}.  </div>  <!-- end: footer.tmpl --> |

А вот главный PHP скрипт:

|  |
| --- |
| <?php  // формируем массив  $nav = array(  'primary' => array(  array('name' => 'Clothes', 'url' => '/clothes'),  array('name' => 'Shoes and Accessories', 'url' => '/accessories'),  array('name' => 'Toys and Gadgets', 'url' => '/toys'),  array('name' => 'Books and Movies', 'url' => '/media'),  ),  'secondary' => array(  array('name' => 'By Price', 'url' => '/selector/v328ebs'),  array('name' => 'By Brand', 'url' => '/selector/gf843k2b'),  array('name' => 'By Interest', 'url' => '/selector/t31h393'),  array('name' => 'By Recommendation', 'url' => '/selector/gf942hb')  )  );    include 'Twig/Autoloader.php';  Twig\_Autoloader::register();    try {  $loader = new Twig\_Loader\_Filesystem('templates');    $twig = new Twig\_Environment($loader);    $template = $twig->loadTemplate('shop.tmpl');    echo $template->render(array (  'nav' => $nav,  'updated' => '24 Jan 2011'  ));    } catch (Exception $e) {  die ('ERROR: ' . $e->getMessage());  }  ?> |

Обратите внимание, что нам не нужно загружать все шаблоны функцией loadTemplate. Главное подключить основной шаблон. Каждый мелкий под-шаблон загрузится автоматом. Переменные и значения, переданные в главный шаблон, будут доступны во всех подключаемых шаблонах.



Фильтрация данных

Мы уже затронули тему фильтров. Давайте посмотрим, какие ещё возможности в данной сфере предоставляет нам Twig.

Давайте рассмотрим, к примеру, фильтр ‘**date’**. Данный фильтр даёт нам возможность формировать дату и время, используя нативные для PHP маркеры. Пример:

|  |
| --- |
| <html>  <head></head>  <body>  {{ "now"|date("d M Y h:i") }} <br/>  {{ "now"|date("d/m/y") }}  </body>  </html> |

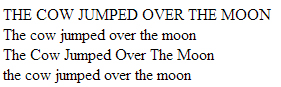
Результат:



Также вы можете воспользоваться фильтрами `**upper**`, `**lower**`, `**capitalize**`, `**title**` для контроля заглавных и прописных букв:

|  |
| --- |
| <html>  <head></head>  <body>  {{ "the cow jumped over the moon"|upper }} <br/>  {{ "the cow jumped over the moon"|capitalize }} <br/>  {{ "the cow jumped over the moon"|title }} <br/>  {{ "The Cow jumped over the Moon"|lower }} <br/>  </body>  </html> |

Результат:



Фильтр `**striptags**` уберёт из текста все HTML и XML элементы:

|  |
| --- |
| <html>  <head></head>  <body>  {{ "<div>I said \"<b>Go away!</b>\"</div>"|striptags }} <br/>  </body>  </html> |

Результат



Фильтр `**replace**` позволяет быстро и просто заменять какие-то значения в строке на нужные нам. Пример:

|  |
| --- |
| <html>  <head></head>  <body>  {{ "I want a red boat"|replace({"red" : "yellow", "boat" : "sports car"}) }} <br/>  </body>  </html> |



Вы уже видели фильтр `**escape**` в действии. В Twig также есть фильтр, который делает абсолютно противоположное действие - `**raw**`. Его следует использовать только для html кода, который вы считаете 100% безопасным.

|  |
| --- |
| <html>  <head></head>  <body>  Escaped output: {{ html|escape }} <br/>    Raw output: {{ html|raw }} <br/>  </body>  </html> |

Если же вам нужно применить `escape` к большому блоку кода, то вы можете воспользоваться синтаксисом `**autoescape**`, передав булево значение true/false для активации и дезактивации фильтрации `escape`. Пример:

|  |
| --- |
| <html>  <head></head>  <body>  {% autoescape true %}  Escaped output: {{ html }} <br/>  {% endautoescape %}    {% autoescape false %}  Raw output: {{ html }}  {% endautoescape %}  </body>  </html> |

Теперь вы уже больше знаете о Twig-e и можете использовать условия, циклы и фильтры.

Исключения (Exceptions)

В примерах с подключением библиотек Twig Вы увидели новую для себя конструкцию try…catch. Она непосредственная связана с одной крайне важной с точки зрения архитектуры сущностью – исключениями.

**Исключения** являются специальными условиями, обычно в случае ошибки, которые проявляются или могут быть специально созданы программой. Они указывают на то, что что-то в процессе отличается от предполагаемого хода событий. В большинстве случаев подобные ситуации требуют выполнения специальных инструкций, чтобы предотвратить неконтролируемое завершение процесса.

Исключения могут быть брошены (thrown) и пойманы (caught). Когда исключение брошено, то значит, произошло что-то выходящее за рамки нормального течения процесса работы программы, и требуется выполнение какой-то другой функции. Подхват исключения осуществляется в специальной функции, которая сообщает остальной программе о том, что она готова обработать исключение.

Сформировать обработку исключений в PHP очень просто. Сами исключения являются частью стандартной библиотеки PHP SPL, которая входит в в ядро PHP, начиная с версии 5, и не требует дополнительного подключения чего-либо.

Исключения реализуются так же как и любой другой объект:

|  |
| --- |
| <?php  $exception = new Exception();  throw $exception;  ?> |

Также, как и любой объект, они имеют методы, которые можно вызвать. Данные методы облегчают формирование реакции на исключение. Рассмотрим список методов у данного класса. Ниже список методов исключений:

* getMessage() – получает сообщение исключения;
* getCode() – возвращает числовой код, который представляет исключение;
* getFile() – возвращает файл, в котором произошло исключение;
* getLine() – возвращает номер строки в файле, где произошло исключение;
* getTrace() – возвращает массив backtrace() до возникновения исключения;
* getPrevious() – возвращает исключение, произошедшее перед текущим, если оно было;
* getTraceAsString() – возвращает массив backtrace() исключения в виде строки;
* \_\_toString() –возвращает все исключение в виде строки. Данную функцию можно переписать.

Исключения хороши тем, что это стандартизованный механизм. Таким образом, человеку, не работавшему с вашим кодом, не нужно будет читать мануал, чтобы понять, как обрабатывать ошибки. Ему достаточно знать, как работают исключения. При этом с исключениями гораздо проще находить источник ошибок, так как всегда есть стек вызовов (trace).

Если какое-либо из действий выбрасывает исключение, то его можно поймать в месте выполнения данного кода. Для этого нужно обернуть выполняемый блок в конструкцию try…catch

|  |
| --- |
| <?php  class A{  function myMethod(){  // do something  throw new Exception("Exception time!");  }  }  ?> |
| <?php  $my\_a = new A();  try{  $my\_a-> myMethod();  }  catch(Exception $e){  echo $e->getMessage();  }  ?> |

В данном примере мы выбрасываем исключение в методе класса A. При работе с ним мы берём его в блок try…catch. Если этого не сделать, то в случае выброса исключения мы получим ошибку Unhandled exception (Необработанное исключение). Обычно IDE отслеживают возможность выброса исключений и подсказывают, когда нужно обернуть тот или иной блок в try…catch.

В наших примерах исключения обрабатываются достаточно грубо – мы выключаем скрипт, но даём понять, что именно и где пошло не так. Обычно исключение логируется в специальное хранилище для последующего изучения причин и их устранения.

Исключения могут наследоваться, как и другие классы. В идеале у вас должен быть базовый класс исключений, от которого наследуются все исключения, бросаемые в вашем коде. Блоков catch может быть много, по одному на каждый класс перехватываемых исключений. Таким образом можно создать фильтр исключений, т.е. перехватывать не все, а только избранные типы исключений, а все остальные будут перехвачены стандартным обработчиком PHP.

В PHP 5.5 и более поздних версиях также можно использовать блок **finally** после или вместо блока catch. Код в блоке finally всегда будет выполняться после кода в блоках try и catch, вне зависимости было ли брошено исключение или нет, перед тем как продолжится нормальное выполнение кода.

Хорошим тоном при создании сайтов считается полное отсутствие ошибок. Если же ошибки возникают - они должны быть обработаны и представлены пользователю и разработчику в понятном виде.

# Домашнее задание

1. Создать галерею изображений, состоящую из двух страниц:
   1. Просмотр всех фотографий (уменьшенных копий);
   2. Просмотр конкретной фотографии (изображение оригинального размера)
   3. Все страницы вывода на экран - это twig-шаблоны. Вся логика - на бэкенде.
   4. (со звёздочкой) Реализовать хранение ссылок и информации по картинкам в БД
2. (со звёздочкой) Для примера 6 из сегодняшнего урока реализовать хранение в БД, которое позволит логике example6.php работать.

# Дополнительные материалы и литература

1. http://ruseller.com/lessons.php?rub=37&id=1652 – Twig в примераз
2. http://ruseller.com/lessons.php?rub=37&id=1277 – Исключения в примерах
3. https://github.com/fabpot/Twig/ - Twig исходники
4. https://habrahabr.ru/post/100137/ - как правильно применять исключения
5. Мэтт Зандстра - "PHP. Объекты, шаблоны и методики программирования"
6. http://php.net/manual/ru/language.exceptions.php - исключения (exceptions) в PHP