МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ГЛАВНОЕ УПРАВЛЕНИЕ ПО ОБРАЗОВАНИЮ

МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

|  |  |
| --- | --- |
| Специальность | 2– 40 01 01 |
| Учебная группа | ПО-455 |
|  |  |

Учебная дисциплина Программные средства

создания интернет-

приложений

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №24**

**ВНЕДРЕНИЕ PHP-СЦЕНАРИЕВ В HTML-ДОКУМЕНТ**

Выполнил Савич А.О.

Проверил Денисовец Д. А.

2022

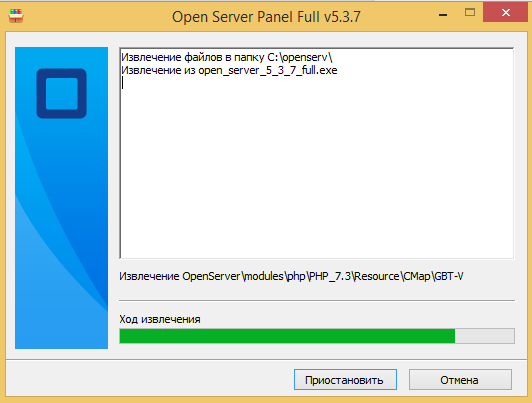
**1 Цель работы**

1.1 Формирование умений внедрять PHP-сценарии в HTML-документ

**2 Индивидуальное задание**

**Вариант 8**

1 Установите локальный сервер Denwer. Настройте локальный сервер Denwer.



2 Создайте свой новый виртуальный хост. Создайте некоторый простейший сайт (Автомобили).

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">

<title>Âûâîä äàííûõ</title>

</head>

<body>

<fieldset>

<form method="post" action="select\_user.php">

<label for="first\_name">Название автомобиля</label>

<br/>

<input type="text" name="first\_name" size="30"><br/>

<input id="submit" type="submit" value="Начать поиск"><br/>

</fieldset>

<fieldset>

</fieldset>

</body>

</html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">

<title>results</title>

</head>

<body>

<?php

$connt = mysqli\_connect("localhost", "root", "root","users");

$first\_name = trim($\_REQUEST['first\_name']);

$sql\_select = "SELECT \* FROM users WHERE topic='$first\_name'";

$result = mysqli\_query($connt,$sql\_select);

while ($row = mysqli\_fetch\_array($result)) {

$field1name = $row["keysp"];

$field2name = $row["date"];

$field3name = $row["topic"];

echo '<table border="1" cellspacing="5" cellpadding="5" width="100%"><tr>

<td>'.$field1name.'</td>

<td>'.$field2name.'</td>

<td>'.$field3name.'</td>

<td>'.$field8name.'</td>

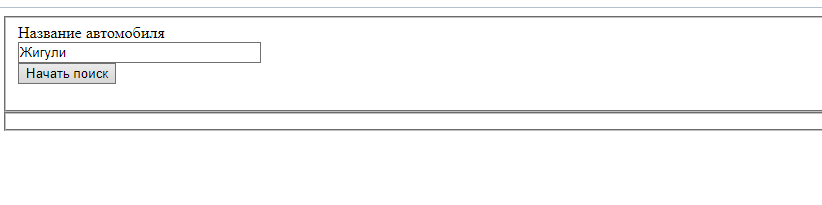
</tr></table>';

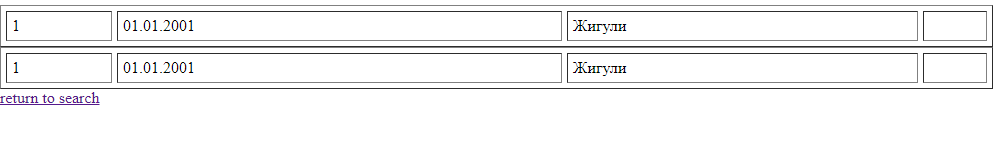
}

?>

<a href="zadanie3.html">return to search</a><br/><br/>

</body>

</html>



3 Контрольные вопросы

1 Объясните способы проверки данных, введенных пользователем, в языке программирования PHP.

## Что такое валидация?

Валидация означает проверку данных, вводимых пользователем. В PHP доступны два типа проверки:

* **Валидация на стороне клиента** — проверка выполняется на стороне клиента в веб-браузере.
* **Валидация на стороне сервера** — после отправки данных на сервер их проверка осуществляется на серверной стороне.

Ниже приведена HTML-форма, которая содержит различные поля ввода: обязательные (required) и необязательные текстовые поля, переключатели (радио-кнопки) и кнопку отправки (submit). С этой формой мы будем с вами работать в этой главе. Попробуйте ввести данные:

### Некоторые из правил проверки нашей формы:

|  |  |
| --- | --- |
| **Поле формы** | **Правила валидации** |
| Имя | Обязатеьно для заполнения + Должно содержать только буквы и пробелы |
| E-mail | Обязатеьно для заполнения + Требуются символ собачка **(@)** и точка **(.)** |
| Веб-сайт | Необязательно для заполнения. Проверяется наличие валидного URL |
| Комментарий | Необязательно для заполнения. Многострочное поле ввода (текстовое поле) |
| Образование | Обязатеьно для заполнения + Должна быть выбрана хотя бы одна кнопка |

## Текстовые поля

Поля для ввода имени, адреса электронной почты и веб-сайта создается с помощью элемента **<input>** (от англ. input — ввод) с атрибутом type="text", а поле для комментария применяется элемент **<textarea>** (текстовая область). HTML-код выглядит так:

Имя: <input type="text" name="name">  
E-mail: <input type="text" name="email">  
Веб-сайт: <input type="text" name="website">  
Комментарий: <textarea name="comment" rows="5" cols="40"></textarea>

## Радио-кнопки

В нашей форме выбор образования осуществляется с помощью элементов <input> типа **radio** (переключатели), которые используют принцип логического «ИЛИ», позволяя выбрать только одно из нескольких значений: если вы выбираете одно положение, то все остальные становятся неактивными:

<input type="radio" name="education" value="Институт">Институт  
<input type="radio" name="education" value="Техникум">Техникум  
<input type="radio" name="education" value="Другое"Другое>

## Элемент формы (form)

Главным для элемента <form> является атрибут **action**, который указывает обработчик данных для формы. Обработчик данных — это файл, описывающий, что нужно делать с данными формы. Данные формы отправляются с помощью method = "post":

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

$\_SERVER ["PHP\_SELF"] — это суперглобальная переменная, которая возвращает имя файла текущего выполняемого скрипта-обработчика.

Переменная $\_SERVER ["PHP\_SELF"] отправляет данные из формы на саму же страницу с формой, вместо перехода на другую страницу. Таким образом, пользователь будет получать сообщения об ошибках на той же странице, где заполняется форма.

Функция htmlspecialchars() преобразует данные, введенные пользователем, которые могут содержать нежелательные HTML-тэги. Производятся следующие преобразования:

* '&' (амперсанд) преобразуется в '&amp;'
* '<' (знак "меньше чем") преобразуется в '&lt;'
* '>' (знак "больше чем") преобразуется в '&gt;'

Эти манипуляции предотвращает использование злоумышленниками кода путем внедрения скрипта (атаки с межсайтовым скриптингом) в формы.

## Примечание о безопасности форм PHP

Учтите, что переменная $\_SERVER ["PHP\_SELF"] может использоваться хакерами!

Если Вы используете на странице сайта PHP\_SELF, то пользователь может ввести в адресной строке косую черту (/), а затем выполнить несколько команд межсайтового скриптинга (XSS).

**Примечание:** XSS (англ. Cross-Site Scripting — «межсайтовый скриптинг») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. XSS позволяет злоумышленникам внедрять клиентские скрипты в веб-страницы, просматриваемые другими пользователями..

Предположим, у нас есть следующая форма на странице с именем "send\_form.php":

<form method="post" action="<?php echo $\_SERVER["PHP\_SELF"];?>">

Теперь, если пользователь вводит обычный URL-адрес в адресной строке, например «http://site\_name.com/send\_form.php», приведенный выше код будет преобразован в:

<form method="post" action="send\_form.php">

Теперь пользователь вводит URL-адрес в адресной строке и после косой черты несколько команд межсайтового скриптинга:

http://site\_name.com/send\_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

После таких манипуляция приведенный выше код будет переведен на:

<form method="post" action="send\_form.php/"><script">('hacked')</script">

Этот код добавляет тег скрипта и команду предупреждения alert. Когда страница начнёт загружаться, код JavaScript будет выполнен и пользователь увидит окно предупреждения. Дальнейшая загрузка страницы прекратится до тех пор, пока пользователь не кликнет ОК. Это простой пример того, как можно использовать переменную PHP\_SELF.

**Примечание:** Имейте в виду, что любой код JavaScript можно внедрить внутрь тега ><script">! Хакер может перенаправить посетителя сайта к файлу на другом сервере, а тот файл может содержать вредоносный код. Этот код, в свою очередь, может изменять глобальные переменные или пренаправлять данные формы на другой адрес, например, для кражи пользовательской информации.

## Как избежать эксплойтов $\_SERVER["PHP\_SELF"]?

Экспло́йт (англ. exploit, эксплуатировать) — это фрагмент вредоносного программного кода либо последовательность команд, которые используют уязвимости в программе и применяеются для проведения хакерской атаки.

Чтобы избежать экспойты нужно переменную $\_SERVER["PHP\_SELF"] предать в качестве аргумента в функцию htmlspecialchars():

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

Функция htmlspecialchars() преобразует специальные символы (в нашем случае угловые скобки < и >) в объекты HTML. Теперь, если пользователь попытается использовать переменную PHP\_SELF с целью внедрения стороннего кода, это приведет к следующему выводу:

<form method="post" action="send\_form.php/&quot;&gt;&lt;script&quot;&gt;('hacked')&lt;/script&gt;">

Попытка эксплойта в этом случае не удастся и никакой вредоносный код не сработает.

## Проверка данных формы с помощью PHP

Первым делом передадим все переменные формы в функцию PHP htmlspecialchars(). Эта функция возвращает строку, над которой проведены рассмотренные выше преобразования. Этих преобразований достаточно для защиты от эксплойта.

Теперь код можно безопасно отображать на странице или в электронном письме.

Сдедующим шагом применим ещё две функции:

1. Из данных, вводимых пользователем (с помощью PHP функции trim()) удалим ненужные символы (лишние пробелы, табуляции, переходы на новую строку)
2. C помощью PHP функции stripslashes() из данных, вводимых пользователем, удалим обратную косую черту **(\)**

И, наконец, создадим функцию, которая будет выполнять все рассмтренные проверки за нас, что намного удобнее, чем писать один и тот же код снова и снова. Назовем функцию test\_input().

В следующем примере будем проверять каждую переменную $\_POST с помощью функции test\_input():

#### Пример

[Попробуй сам »](https://www.wm-school.ru/php/edit/exercise109.php)

<?php

//определим переменные и установим пустые значения

$name = $email = $education = $comment = $website = "";

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$name = test\_input($\_POST["name"]);

$email = test\_input($\_POST["email"]);

$website = test\_input($\_POST["website"]);

$comment = test\_input($\_POST["comment"]);

$education = test\_input($\_POST["education"]);

}

function test\_input($data) {

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data);

return $data;

}

?>

В начале сценария мы проверяем, используя суперглобальную переменную $ \_SERVER ["REQUEST\_METHOD"], была ли отправлена ​​форма. Если для запроса страницы был использован метод POST, значит, форма была отправлена ​​— и ее нужно проверить. Если форма не была отправлена — пропускаем проверку и отображаем пустую форму.

В приведенном выше примере все поля ввода необязательны.

2 Охарактеризуйте абсолютный URL адрес в языке программирования PHP.

В PHP существует такое же разделение. С одной стороны - реальная файловая система с реальными папками и файлами, с другой - URL адреса, которые могут не иметь ничего общего с реальной файловой структурой.

Если ты видишь на каком-нибудь сайте страницу с URL /category/monitors - это совсем не значит, что на сайте есть скрипт /category/monitors/index.php.

Вполне вероятно, что и такой папки там тоже нет, а все URL адреса обрабатываются одним единственным PHP файлом.

И даже если в URL присутствует расширение файла, например /about.html - это тоже не говорит о существовании файла about.html. Может он есть, а может и нет.

Новички часто путают пути к реальным файлам с URL адресами. Сейчас я разберу пару таких ошибок, на примере которых можно будет прочувствовать разницу.

## Абсолютный путь в PHP

Абсолютный путь - это полный путь к папке или файлу. Вот пара примеров для разных операционных систем:

* D:\OpenServer\domains\test.ru\index.php - для OpenServer на Windows
* /var/www/html/test.ru/index.php - для Ubuntu

Как видите, это полный путь от корня диска до конкретного файла или папки. Начинается со слеша или буквы диска (Windows).

Получить абсолютный путь скрипта можно с помощью магической константы \_\_FILE\_\_:

<?php

echo \_\_FILE\_\_;

Для получения абсолютного пути к папке, в которой находится скрипт, есть магическая константа \_\_DIR\_\_:

<?php

echo \_\_DIR\_\_;

Как этим пользоваться. Допустим, у нас в корне сайта лежат файлы index.php и config.php и мы хотим подключить второй в первый.

Если мы хотим подключить config.php по его абсолютному пути, есть два способа сделать это:

<?php

# Указать путь вручную

require 'D:\OpenServer\domains\test.ru\cpnfig.php';

# Подставить путь к текущей папке и добавить имя файла вручную

require \_\_DIR\_\_ . '/config.php';

Поскольку константа \_\_DIR\_\_ не добавляет слеш после последней папки, мы указываем его вручную.

## Относительный путь в PHP

У относительных путей в PHP есть один подвох - они могут строиться относительно не той папки, от которой мы ожидаем.

Дело в том, что когда мы подключаем скрипт по относительному пути require('config.php');, PHP будет искать его по следующему алгоритму:

* Сначала PHP попытается найти этот файл в папках, указанных в директиве include\_path. Посмотреть, что указано в этой директиве конкретно у вас можно с помощью var\_dump(get\_include\_path());, папки разделяются символом ;

Если мы укажем путь к скрипту в таком виде: require('./config.php');, то этот шаг будет пропущен.

* Затем PHP попытается найти файл в папке, в которой находится подключающий скрипт (в нашем случае index.php).
* Далее PHP попытается найти файл в папке текущего рабочего каталога.

Например, если мы в index.php подключили файл scripts/script.php, а в этом самом script.php уже пытаемся подключить файл по относительному пути, тогда поиск файла произойдёт и в папке scripts тоже.

Два последних пункта - главная проблема относительных путей. Мы не можем быть уверены, что будет подключен именно тот файл, который нам нужен.

Именно по этой причине я призываю тебя **отказаться от использования относительных путей в PHP**.

Кому-то из практикующих разработчиков эта фраза может не понравиться, но я считаю это единственным разумным решением.

Тем более нет ничего сложного в добавлении константы \_\_DIR\_\_ перед именем скрипта, что автоматически сделает путь абсолютным.

<?php

# Отказываемся

require 'config.php';

# Используем

require \_\_DIR\_\_ . '/config.php';

## Абсолютный путь в URL

Абсолютный путь в URL означает путь **от корня сайта**. Корень сайта - это папка, которая содержит публичную часть сайта, т.е. доступную извне.

По-умолчанию корень сайта - это сама папка с сайтом, например site.ru, но зачастую её меняют на site.ru/public, чтобы часть скриптов нельзя было запустить по прямой ссылке в браузере.

Есть несколько способов указать абсолютный путь в URL:

* http://test.ru/article.php - полный путь с протоколом и доменом
* //test.ru/article.php - полный путь без протокола (берётся из текущей страницы). Содержит два слеша в начале.
* /article.php - полный путь без протокола и домена (берутся из текущей страницы). Содержит слеш в начале.

Второй и третий варианты удобны тем, что при миграции с http на https и обратно все ссылки автоматически сменят протокол, не нужно будет бегать по всему сайту и менять вручную.

Лично я практически всегда использую третий вариант, кроме случаев, когда нужно указать ссылку на другой поддомен (blog.site.ru, shop.site.ru и т.д.).

## Относительный путь в URL

Относительные пути в URL указываются без слеша в начале ссылки, например:

<a href="articles">Ссылка</a>

Относительные пути в URL более предсказуемы, чем в PHP. Но я рекомендую использовать их только там, где это действительно необходимо.

Чаще всего их использование приводит к путанице. И вот пара типичных проблем, с которыми часто сталкиваются новички.

## Текущий и родительский каталоги

Помимо указания конкретных папок, мы также можем добавить в путь указание "перейти на папку выше", например:

<?php

include '../config.php';

В коде выше мы подключим файл config.php, который находится не в текущей папке, а в родительской. С абсолютными путями это тоже работает:

<?php

include \_\_DIR\_\_ . '/../config.php';

И с URL-адресами тоже:

<a href="../article.php">Статья</a>

Также мы можем указать ссылку на текущий каталог, что бывает актуально в некоторых операционных системах:

<?php

include './config.php';

3 Охарактеризуйте относительный URL адрес в языке программирования PHP.

В PHP существует такое же разделение. С одной стороны - реальная файловая система с реальными папками и файлами, с другой - URL адреса, которые могут не иметь ничего общего с реальной файловой структурой.

Если ты видишь на каком-нибудь сайте страницу с URL /category/monitors - это совсем не значит, что на сайте есть скрипт /category/monitors/index.php.

Вполне вероятно, что и такой папки там тоже нет, а все URL адреса обрабатываются одним единственным PHP файлом.

И даже если в URL присутствует расширение файла, например /about.html - это тоже не говорит о существовании файла about.html. Может он есть, а может и нет.

Новички часто путают пути к реальным файлам с URL адресами. Сейчас я разберу пару таких ошибок, на примере которых можно будет прочувствовать разницу.

## Абсолютный путь в PHP

Абсолютный путь - это полный путь к папке или файлу. Вот пара примеров для разных операционных систем:

* D:\OpenServer\domains\test.ru\index.php - для OpenServer на Windows
* /var/www/html/test.ru/index.php - для Ubuntu

Как видите, это полный путь от корня диска до конкретного файла или папки. Начинается со слеша или буквы диска (Windows).

Получить абсолютный путь скрипта можно с помощью магической константы \_\_FILE\_\_:

<?php

echo \_\_FILE\_\_;

Для получения абсолютного пути к папке, в которой находится скрипт, есть магическая константа \_\_DIR\_\_:

<?php

echo \_\_DIR\_\_;

Как этим пользоваться. Допустим, у нас в корне сайта лежат файлы index.php и config.php и мы хотим подключить второй в первый.

Если мы хотим подключить config.php по его абсолютному пути, есть два способа сделать это:

<?php

# Указать путь вручную

require 'D:\OpenServer\domains\test.ru\cpnfig.php';

# Подставить путь к текущей папке и добавить имя файла вручную

require \_\_DIR\_\_ . '/config.php';

Поскольку константа \_\_DIR\_\_ не добавляет слеш после последней папки, мы указываем его вручную.

## Относительный путь в PHP

У относительных путей в PHP есть один подвох - они могут строиться относительно не той папки, от которой мы ожидаем.

Дело в том, что когда мы подключаем скрипт по относительному пути require('config.php');, PHP будет искать его по следующему алгоритму:

* Сначала PHP попытается найти этот файл в папках, указанных в директиве include\_path. Посмотреть, что указано в этой директиве конкретно у вас можно с помощью var\_dump(get\_include\_path());, папки разделяются символом ;

Если мы укажем путь к скрипту в таком виде: require('./config.php');, то этот шаг будет пропущен.

* Затем PHP попытается найти файл в папке, в которой находится подключающий скрипт (в нашем случае index.php).
* Далее PHP попытается найти файл в папке текущего рабочего каталога.

Например, если мы в index.php подключили файл scripts/script.php, а в этом самом script.php уже пытаемся подключить файл по относительному пути, тогда поиск файла произойдёт и в папке scripts тоже.

Два последних пункта - главная проблема относительных путей. Мы не можем быть уверены, что будет подключен именно тот файл, который нам нужен.

Именно по этой причине я призываю тебя **отказаться от использования относительных путей в PHP**.

Кому-то из практикующих разработчиков эта фраза может не понравиться, но я считаю это единственным разумным решением.

Тем более нет ничего сложного в добавлении константы \_\_DIR\_\_ перед именем скрипта, что автоматически сделает путь абсолютным.

<?php

# Отказываемся

require 'config.php';

# Используем

require \_\_DIR\_\_ . '/config.php';

## Абсолютный путь в URL

Абсолютный путь в URL означает путь **от корня сайта**. Корень сайта - это папка, которая содержит публичную часть сайта, т.е. доступную извне.

По-умолчанию корень сайта - это сама папка с сайтом, например site.ru, но зачастую её меняют на site.ru/public, чтобы часть скриптов нельзя было запустить по прямой ссылке в браузере.

Есть несколько способов указать абсолютный путь в URL:

* http://test.ru/article.php - полный путь с протоколом и доменом
* //test.ru/article.php - полный путь без протокола (берётся из текущей страницы). Содержит два слеша в начале.
* /article.php - полный путь без протокола и домена (берутся из текущей страницы). Содержит слеш в начале.

Второй и третий варианты удобны тем, что при миграции с http на https и обратно все ссылки автоматически сменят протокол, не нужно будет бегать по всему сайту и менять вручную.

Лично я практически всегда использую третий вариант, кроме случаев, когда нужно указать ссылку на другой поддомен (blog.site.ru, shop.site.ru и т.д.).

## Относительный путь в URL

Относительные пути в URL указываются без слеша в начале ссылки, например:

<a href="articles">Ссылка</a>

Относительные пути в URL более предсказуемы, чем в PHP. Но я рекомендую использовать их только там, где это действительно необходимо.

Чаще всего их использование приводит к путанице. И вот пара типичных проблем, с которыми часто сталкиваются новички.

## Текущий и родительский каталоги

Помимо указания конкретных папок, мы также можем добавить в путь указание "перейти на папку выше", например:

<?php

include '../config.php';

В коде выше мы подключим файл config.php, который находится не в текущей папке, а в родительской. С абсолютными путями это тоже работает:

<?php

include \_\_DIR\_\_ . '/../config.php';

И с URL-адресами тоже:

<a href="../article.php">Статья</a>

Также мы можем указать ссылку на текущий каталог, что бывает актуально в некоторых операционных системах:

<?php

include './config.php';