

# Введение в PHP

- Введение
- Технология клиент-сервер
- Первая программа
- Конструкции echo, print
- Инструкции
- Комментарии

## **Введение**

---

PHP – это широко используемый язык сценариев общего назначения с открытым исходным кодом. Говоря проще, PHP это язык программирования, специально разработанный для написания веб-приложений (сценариев), исполняющихся на веб-сервере.

**К сведению.** PHP — высокоуровневый язык **сценариев** (англ. **script**) — кратких описаний действий, выполняемых системой. Разница между программами и сценариями довольно размыта, а поэтому слова сценарий (скрипт) и программа будут использоваться как синонимы.

Язык программирования PHP относится к технологии **серверного** программирования, так называемой backend разработки. Тем не менее, прежде чем изучать язык, желательно иметь базовое представление о **клиентских** (frontend) технологиях и разработке пользовательского **интерфейса**:

- HTML;
- CSS;
- JavaScript.

Но возможен и другой подход. Вы начинаете путь веб-разработчика с программирования на PHP, изучаете, увлекаетесь процессом,

вдохновляетесь, и уже тогда начинаете понимать, знаний из какого направления frontend разработки вам не хватает. И не важно, в какой последовательности, важно, что проиграть от погружения в удивительный мир серверного программирования нельзя.

## Что такое PHP?

Язык программирования PHP достаточно мощный, чтобы быть в центре системы WordPress, самой большой системы управления контентом. Достаточно серьезный, чтобы на нем была построена социальная сеть Facebook. В тоже время, это достаточно легкий для изучения язык для новичка.

PHP это аббревиатура: **PHP: Hypertext Preprocessor** - гипертекстовый препроцессор.

PHP файлы могут содержать **текст, HTML, CSS, JavaScript, PHP**-код и, по умолчанию, имеют расширение **.php**.

**Важно.** PHP-скрипт **выполняется на сервере**, результат в виде HTML-кода отправляется в браузер. Именно поэтому пользователь запросивший страницу в глобальной сети Интернет, никогда **не увидит PHP-скрипт**, он увидит **результат работы** скрипта (в отличии от JavaScript-скриптов).

Если то, что написано кажется слишком сложным, идем в следующий раздел, разбираемся наглядно.

**Важно.** Если вы никогда прежде не писали серверных сценариев, напоминаю:

- **Все файлы сценариев должны располагаться в корневой директории созданного вами домена.**

Обязательно экспериментируйте, **запускайте предложенные**

**сценарии** файлов example. Пишите свои. Пробуйте.

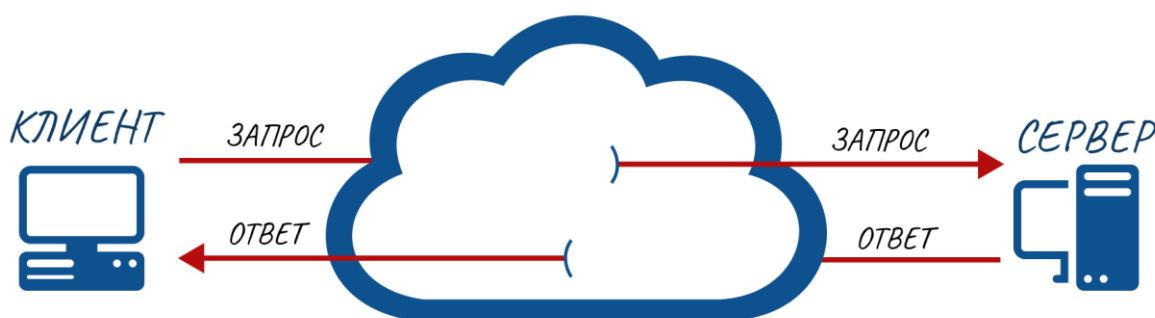
## **Технология клиент-сервер**

---

### **Клиент – сервер**

**Клиент – сервер** – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми **серверами**, и заказчиками услуг, называемыми **клиентами**. Фактически **клиент** и **сервер** – это **программное обеспечение**. Где клиентом может быть терминал в банке или на вокзале, приложение в вашем мобильном устройстве или браузер на вашем компьютере.

Общий принцип клиент-серверных технологий продемонстрирован на **рисунке 1**.



**Рис.1.** Технология клиент-сервер

### **Frontend**

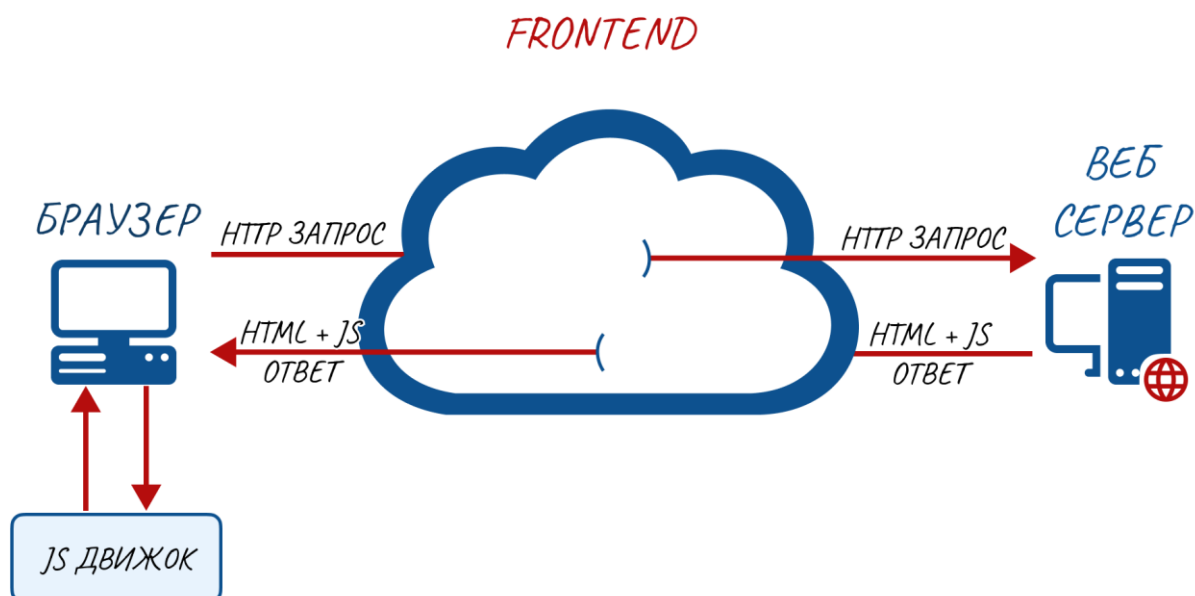
**Frontend** – это разработка пользовательского **интерфейса** и программирование **функций**, которые работают на клиентской стороне веб-сайта или приложения. Frontend-ом называют все, что **видит** пользователь, открывая веб-страницу, и с чем он **взаимодействует**.

Выбор товара в пользовательскую корзину, сортировка карточек, включение / отключение анимации элементов страницы, скрытие и показ панели меню (и многое другое), вот только несколько примеров frontend разработки.

Термин frontend можно перевести как **передняя (видимая)** часть приложения.

**Примечание.** Frontend разработкой мы занимаемся в рамках наших курсов:

- **HTML и CSS.** С нуля до первого сайта.
- **Программирование в JavaScript.** С нуля до клиент-серверных приложений.



**Рис.2.** Frontend разработка

## Backend

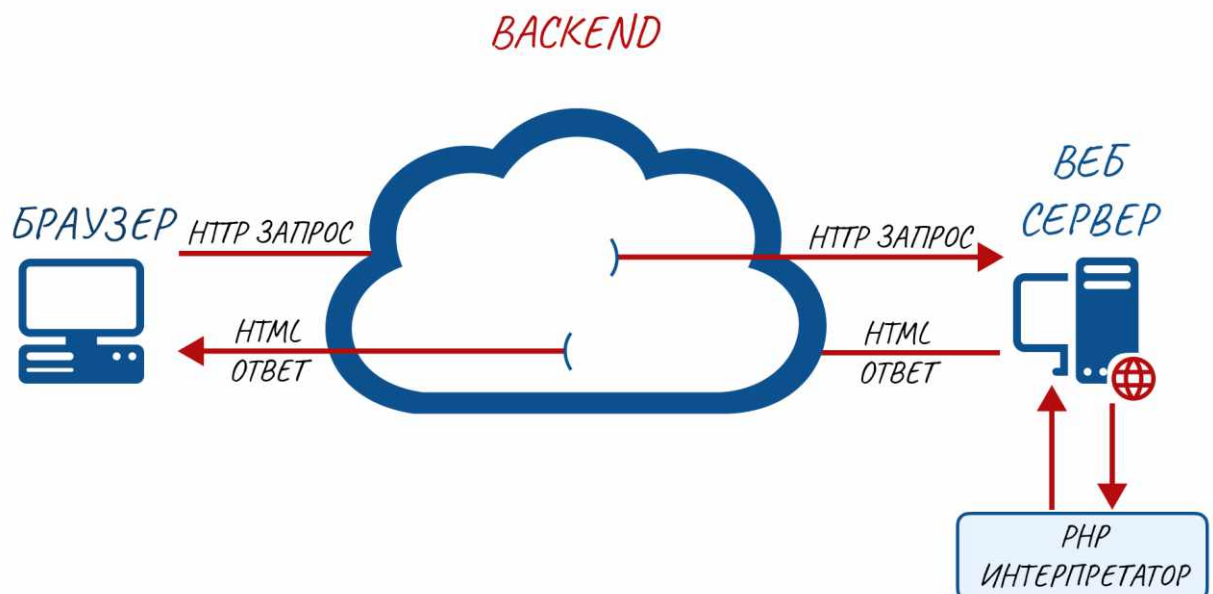
**Backend** – это разработка бизнес логики продукта (сайта или веб-приложения). Backend отвечает за взаимодействие пользователя с внутренними данными, **расположенными на сервере**, которые потом отображает frontend.

Backend – это то, что скрыто от глаз пользователя и **происходит вне его браузера и компьютера** (продолжая логический ряд – вне квартиры, дома, города, иногда страны). Территориально сервер может находиться где угодно.

Термин backend можно перевести как **темная (невидимая)** часть приложения.

Именно **backend** разработке посвящен данный курс.

**Примечание.** Изучив PHP (а рано или поздно это произойдет) не забывайте, язык программирования PHP это не цель, это **инструмент**. Истинной целью backend-разработчика должно быть изучение серверного фреймворка (или двух) и системы создания и управления контентом (CMS – Content Management System). И только тогда вы получите неограниченный доступ к темной стороне разработки ;)



**Рис.3.** Backend разработка

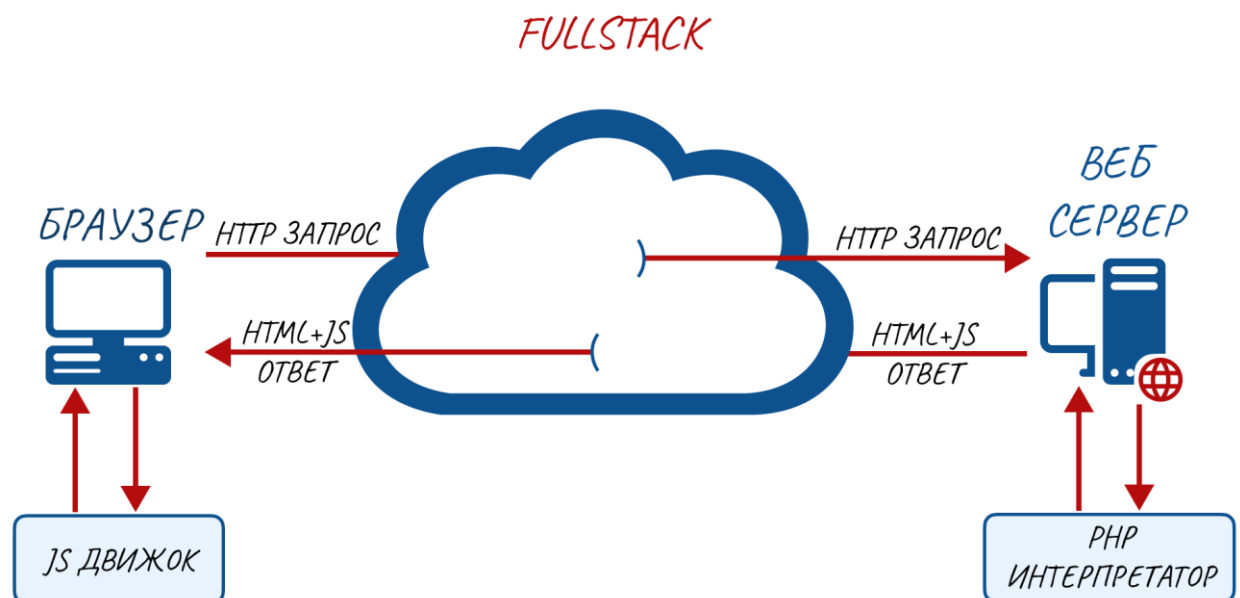
## Fullstack

**Fullstack** – это сфера деятельности разработчика широкого профиля, который умеет работать с фронтендом (программирование на клиентской стороне, пользовательский интерфейс) и бэкендом (серверная сторона программирования приложений и баз данных).

Термин **fullstack** переводится как **полный стек**, так же, как и одноименная структура для хранения данных – **стек**.

**Примечание.** Встретите **fullstack** разработчика, будьте крайне осторожны, и помните, **fullstack** – это белый лебедь мира веб-разработки. Такой же гордый, независимый и умирающий (ИМХО).

Схематично **fullstack** представлен на **рисунке 4**.



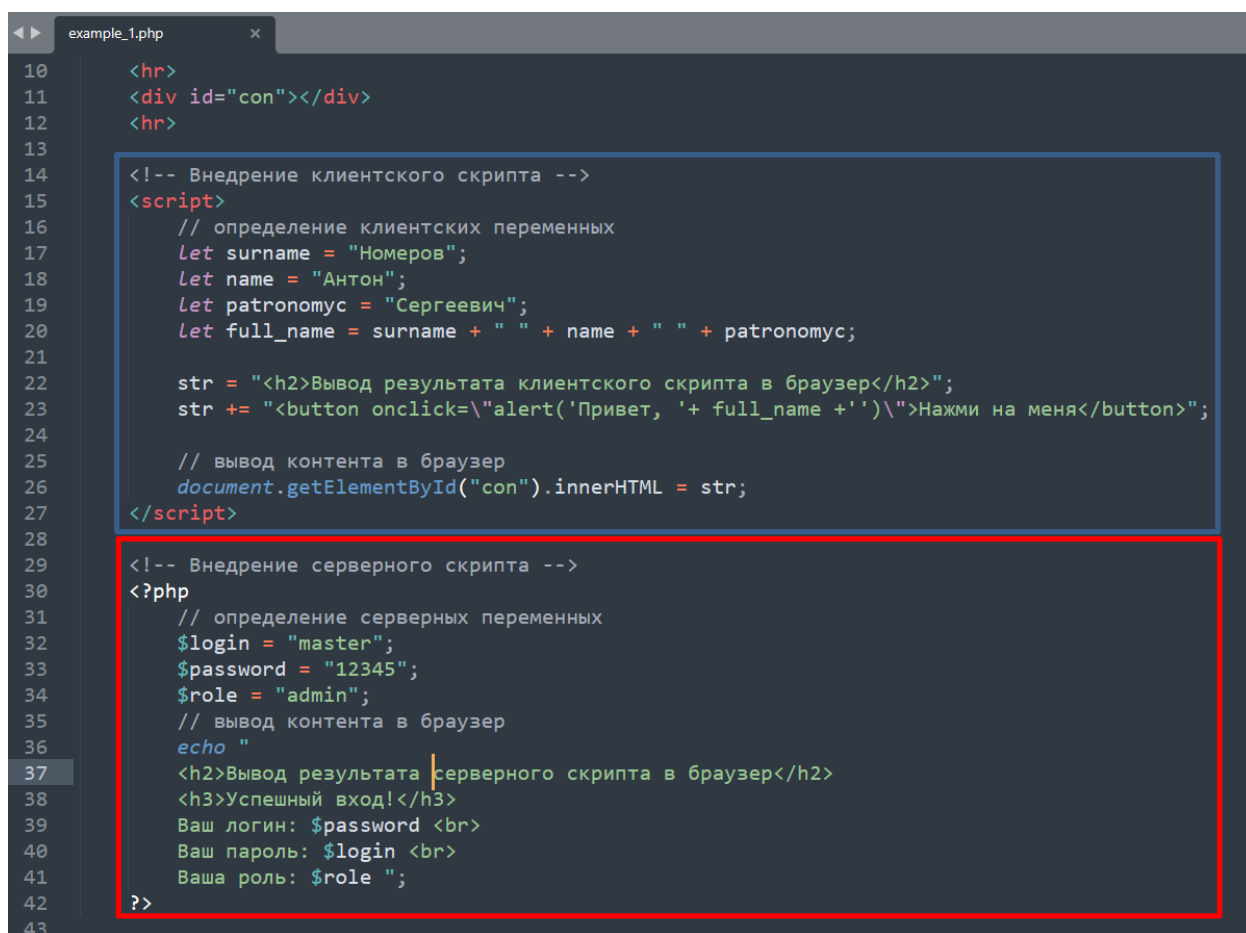
**Рис.4.** Fullstack разработка

Все это так и осталось бы рисунками, если бы не практическая демонстрация нарисованного.

Итак, продемонстрирую сказанное кодом демонстрационного примера. На **рисунке 5** представлен код демонстрационного файла **example\_1**. Файл состоит из HTML-разметки и вставки кодов:

- `<script> ... </script>` - **клиентского** скрипта
- `<?php> ... ?>` - **серверного** скрипта.

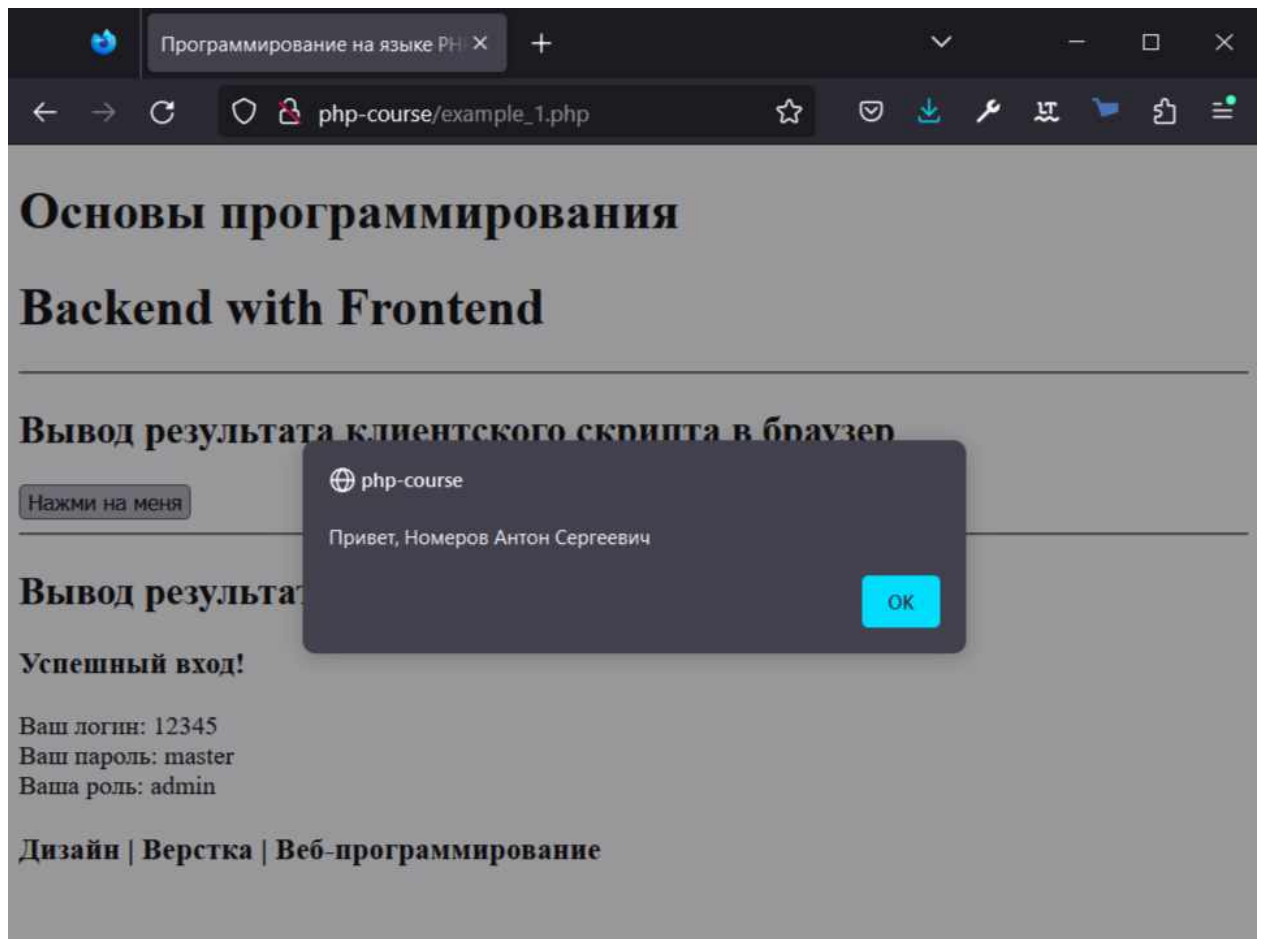
**Примечание.** В предложенном примере ничего понимать не надо (пока)



```
10 <hr>
11 <div id="con"></div>
12 <hr>
13
14 <!-- Внедрение клиентского скрипта -->
15 <script>
16     // определение клиентских переменных
17     let surname = "Номеров";
18     let name = "Антон";
19     let patronomyc = "Сергеевич";
20     let full_name = surname + " " + name + " " + patronomyc;
21
22     str = "<h2>Вывод результата клиентского скрипта в браузер</h2>";
23     str += "<button onclick='\"alert('Привет, ' + full_name + '')\">Нажми на меня</button>";
24
25     // вывод контента в браузер
26     document.getElementById("con").innerHTML = str;
27 </script>
28
29 <!-- Внедрение серверного скрипта -->
30 <?php
31     // определение серверных переменных
32     $login = "master";
33     $password = "12345";
34     $role = "admin";
35     // вывод контента в браузер
36     echo "
37     <h2>Вывод результата серверного скрипта в браузер</h2>
38     <h3>Успешный вход!</h3>
39     Ваш логин: $password <br>
40     Ваш пароль: $login <br>
41     Ваша роль: $role ";
42 ?>
43
```

**Рис.5.** Коды клиентского и серверного скриптов

Загрузите файл в директорию сервера. Оцените вывод браузера.



**Рис.6.** Вывод сценариев примера example\_1 в браузер

На мой взгляд все красиво. Только непонятно пока, что именно демонстрирует этот пример.

Для того чтобы разобраться, чем же именно отличается **клиентское** и **серверное** программирование перейдите в режим разработчика (клавиша F12). Клиентский скрипт присутствует на компьютере пользователя. Кнопочка "Нажми меня" может быть нажата или нет, в зависимости от желания пользователя, в любом случае, клиентский скрипт готов обработать это нажатие.



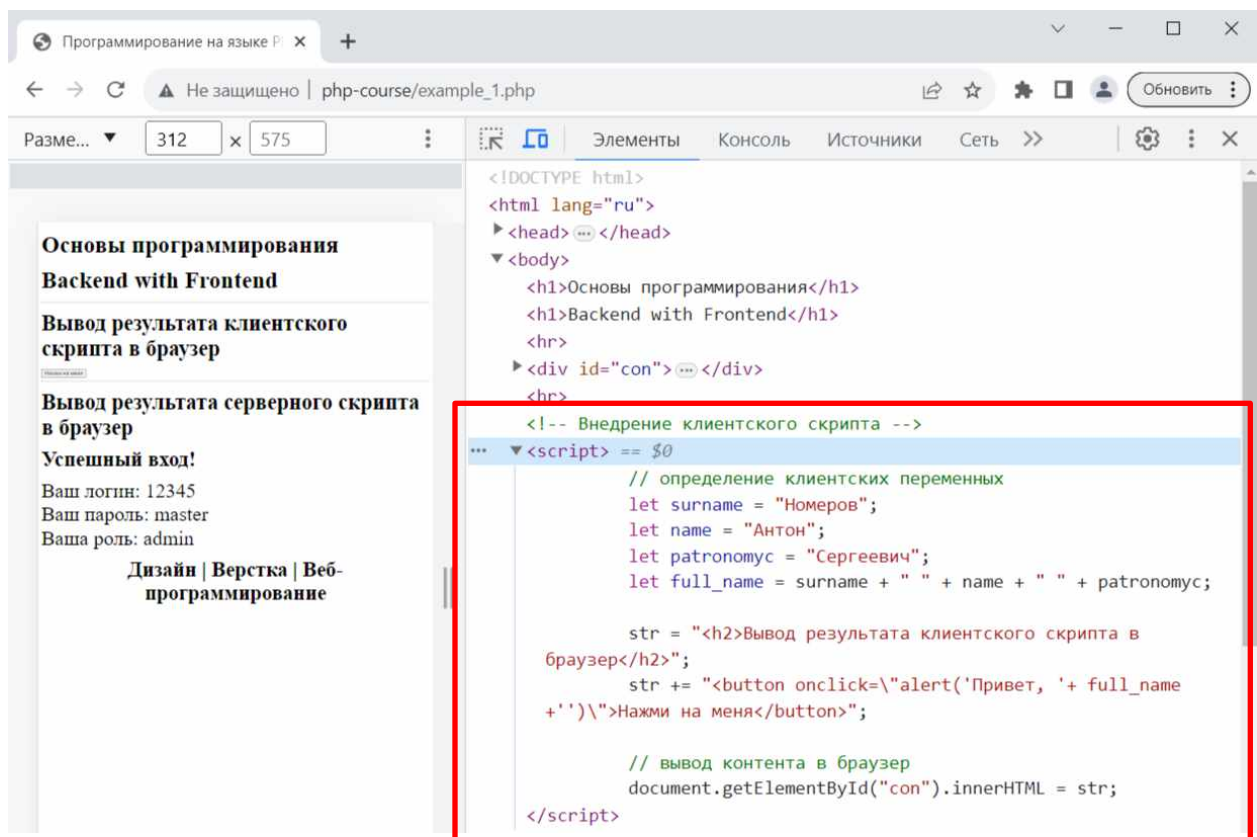


Рис.7. Клиентский скрипт присутствует в коде

А вот **серверного сценария в консоли нет.**

**Важно.** Серверный сценарий давно выполнен **на стороне сервера**. В браузере представлен **результат выполнения скрипта**.

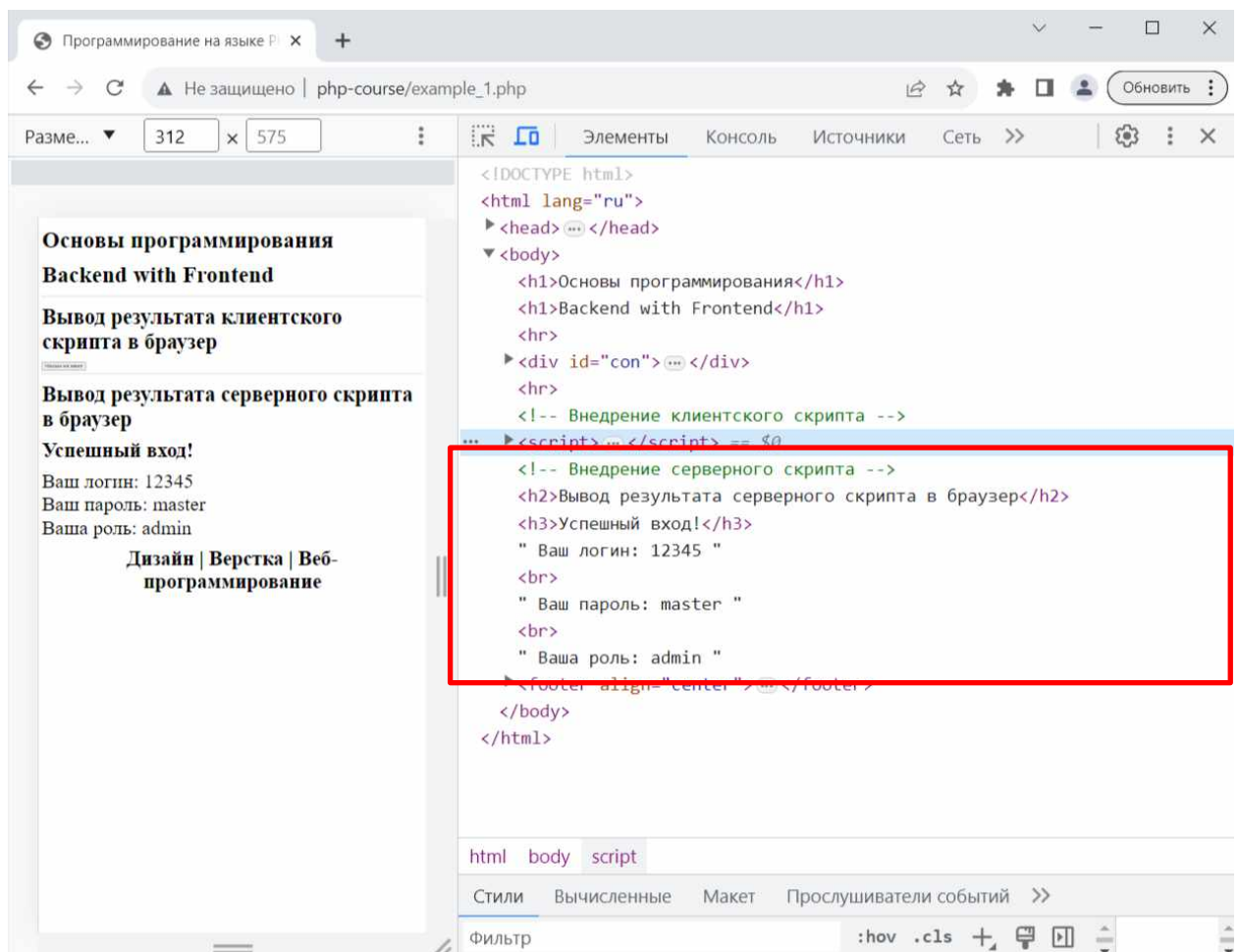


Рис.8. Серверный скрипт давно выполнен

## Первая программа

PHP-скрипт может быть размещен в **любом месте** документа.

При программировании в PHP — нельзя начать писать код сразу. Такой код будет воспринят **интерпретатором** (программой, выполняющей код) как обычный текст.

Любой код на PHP должен быть обернут в конструкцию **<?php ?>**.

- **<?php** — открывающий тег
- **?>** — закрывающий тег.

<?php

```
// PHP код находится здесь
```

```
?>
```

Обычно PHP файл содержит HTML теги и некоторые **инструкции** PHP скриптов. Если файл заканчивается скриптом (или состоит только из скрипта), руководство по оформлению кода позволяет **не указывать закрывающий тег** - `?>`.

Напишем первую программу, которая выведет строку приветствия "Hello, User!". Вывод на экран выполняется с помощью специальной команды **echo**. Ниже представлен пример простого PHP-скрипта, в котором используется встроенная функция echo.

#### **example\_2.** Первый PHP сценарий

```
<!DOCTYPE html>

<html lang="ru">

<head>

    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">

    <title>Программирование на языке PHP</title>

</head>

<body>

    <h1>Основы программирования</h1>

    <h2>Первая программа PHP</h2>

    <?php

        echo "Hello, User!";

    ?>
```

```
</body>
</html>
```

## Конструкции *echo*, *print*

Конструкции **echo** и **print** практически одинаковы. Обе они используются для вывода данных на экран.

Отличия в конструкциях небольшие, но они есть:

- Конструкция **echo** **не имеет** возвращаемого значения, **print** **возвращает значение 1**, которое можно использовать в выражениях.
- Конструкция **echo** может иметь **несколько** параметров (хотя такое использование редко), **print** может принимать **один** аргумент.
- Конструкция **echo** несколько **быстрее**, чем **print**.

Обе конструкции могут использоваться как с круглыми скобками, так и без них: `echo`, `echo()` или `print`, `print()`.

```
<?php
    // вывод в браузер строки

    echo "<h1>Пишем на PHP</h1>";

    // или так

    print("<h1>Первый шаг в серверном программировании</h1>");
?>
```

**Обратите внимание.** Текст может содержать HTML разметку.

В следующем примере показано несколько способов вывода в браузер.

### example\_3. Конструкции вывода

```
<!DOCTYPE html>

<html lang="ru">

<head>

    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">

    <title>Программирование на языке PHP</title>

</head>

<body>

    <h1>Основы программирования</h1>

    <?php

        // print без скобок

        print "Hello, User!<br />";

        // print со скобками

        print("Изучаем PHP<br>")

    ?>

    <!--

    это уже html-комментарий, им я хотел сказать, что
    количество вставок php-тегов не ограничено

    -->

    <?php

        // можно использовать echo с несколькими параметрами

        // хотя применяется такой вывод не часто

        echo "Можно ", "выводить ", " и так...!";

    ?>
```

```
</body>
```

```
</html>
```

В PHP, все ключевые слова (например: if, else, while, echo и т.п.), классы и функции определенные пользователем **не чувствительны к регистру**.

Таким образом, в приведенном примере, все три написания ключевого слова echo являются правильными: echo, ECHO, EcHo.

## Инструкции

---

**Инструкция** — это команда для компьютера выполнить что-то. **Скрипт** (код на PHP) — это набор инструкций, которые, как правило, отделяются друг от друга символом `;`.

Вот пример кода с двумя инструкциями.

```
<?php
    echo "Hello, User!";
    echo "Начинаем изучать язык PHP!";
```

Теоретически инструкции можно написать **последовательно друг за другом** без переноса на новую строку:

```
<?php
    echo "Hello, User!"; echo "Начинаем изучать язык PHP!";
```

Результат на экране будет таким же, но такой код неудобен для чтения, поэтому инструкции чаще всего располагают друг под другом.

Таким образом:

**Интерпретатор** — программа, которая запускает сценарий, выполняя инструкции строго по очереди.

**Скрипт (сценарий)** – набор инструкций, написанных на языке программирования PHP.

**Инструкция** — это единица исполнения, заканчивается точкой с запятой.

Разработчик, должен понимать этот порядок и уметь мысленно разделять программу на независимые части, удобные для анализа.

## ***Комментарии***

---

Кроме кода, в файлах с исходным кодом могут находиться комментарии. Это текст, который не является частью программы, и нужен программистам для пометок. Его единственная цель, быть прочитанным кем-то, кто его видит.

Комментарии можно использовать для того чтобы:

- другие понимали, что вы **делаете**;
- напомнить себе, что вы **сделали**, - большинство программистов имеет опыт возврата к своей программе через некоторое время (иногда весьма продолжительное). Комментарии могут напомнить вам о том, что вы думали, когда писали код.

Комментарии в PHP бывают двух видов:

### **Однострочные комментарии**

Однострочные комментарии начинаются с **//**. После этих символов может следовать любой текст, вся строка не будет анализироваться и исполняться.

Комментарий может занимать всю строку. Если одной строки мало, то создаются несколько однострочных комментариев:

```
<?php
    // это
    // комментарий
    # это тоже комментарий
```

## Многострочные комментарии

Многострочные комментарии начинаются с `/*` и заканчиваются на `*/`.

```
<?php
    /*
    это многострочный блок комментариев
    охватывает несколько строк
    */
```

**example\_4.** PHP поддерживает несколько способов комментирования

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">
    <title>Программирование на языке PHP</title>
</head>
<body>
    <h1>Основы программирования</h1>
    <?php
        // однострочный комментарий
```



```
/*  
    это многострочный блок комментариев  
    охватывает несколько строк  
*/  
  
# это тоже однострочный комментарий  
  
// можно также использовать комментарии, чтобы исключить  
// части строки кода  
  
$x = 5 /* + 15 */ + 5;  
  
echo $x;  
  
?>
```

</body>

</html>

**Комментарий** - текст в коде программы, который не влияет на функциональность, и добавляется программистами для себя и своих коллег.

# Задание 1

=====

В директории раздаточного материала вам предоставлен стандартный шаблон HTML файла. Оформите файл таким образом, чтобы разметка HTML выводилась с помощью **PHP инструкции echo**.

Для выполнения кода не забудьте изменить расширение файла.

**Н.В.** Избегайте появления одних кавычек внутри других. В случае необходимости использовать кавычки, чередуйте **двойные** и **одинарные** кавычки, например так:

```
echo '<html lang="ru">';
```

# Переменные

- Общие понятия
- Изменение переменной
- Именованые переменных
- Выражения в определениях
- Магические числа
- Ошибки при работе с переменными
- Управление ошибками

## Общие понятия

---

Как и в любом другом языке программирования, в PHP существует такое понятие, как переменная.

**Переменная** — это область оперативной памяти, доступ к которой осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных (исключение составляет специальный вид переменных — константа).

**Важно.** Имена всех переменных в PHP должны начинаться со знака **\$** — так интерпретатору значительно легче "понять" и отличить их, например, в строках.

Отличительным преимуществом PHP является то, что в PHP не нужно ни описывать переменные явно, ни указывать их тип. Переменная объявляется **при первом её использовании в программе**, интерпретатор все делает сам.

Ниже приведены некоторые примеры переменных в PHP:

- `$name`
- `$base64`
- `$_site`

- `$e_mail`

При программировании на PHP можно не скупиться на объявление новых переменных. Принципы экономии памяти, которые были актуальны несколько лет назад, сегодня в расчет не принимаются. Однако, при хранении в переменных больших объемов памяти, неиспользуемые переменные лучше удалять.

Можно сказать, что переменные в PHP — это особые объекты, которые **могут содержать в буквальном смысле все, что угодно** (в отличии от констант).

## ***Изменение переменной***

---

Само слово "переменная", говорит о том, что её можно менять. И действительно, с течением времени внутри программы **значения переменных** могут изменяться.

```
<?php

// инициализируем переменную $team

$team = 'Aerosmith';

print($team); // вывод переменной

// пристрастия могут и поменяться ;)

// значение переменной легко изменить

$team = 'Pink Floyd';

print($team); // вывод нового значения переменной
```

Имя переменной осталось тем же, но внутри другие данные.

Переменные - мощная, и в то же время опасная вещь. Никогда нельзя быть точно уверенным, что внутри неё записано, не проанализировав код, который находится перед переменной. Именно этим занимаются

разработчики во время отладки, когда пытаются разобраться, почему программа не работает, или работает не так, как задумано.

Как вы увидите позже, в PHP есть не только переменные. Более того, переменные не так часто используются с целью их менять, намного чаще их используют с целью **хранить в них какие-то данные**.

## **Именованние переменных**

---

Компьютеру без разницы, как мы называем переменные — он железный, но вот программистам — нет. Мы гораздо чаще читаем код, чем пишем. Причём не свой, а написанный другими людьми.

**Важно.** От качества и понятности имён переменных зависит половина успеха **в анализе кода**. По именам переменным легко определить уровень написавшего код программиста.

**Факт.** Всего выделяют три уровня программистов: **Junior, Middle, Senior**. Эти категории присущи всем языкам программирования. Отличием Middle от Junior, в числе прочего, является умение писать грамотный, легко читаемый другими разработчиками код.

Лучше посидеть и придумать название, которое описывает суть, смысл переменной, чем назвать её как попало, а в будущем переделывать. Постарайтесь давать им такие имена, чтобы они были максимально понятны **без контекста**, без изучения окружающего кода.

Для имени переменной используется любой набор допустимых символов, к которым относятся **буквы английского алфавита, цифры**, знак `_`.

**При этом цифру нельзя ставить в начале.**

Имена переменных **регистрозависимы**: например, имя `$hello` и имя `$heLLo` - это два разных имени, а значит и две переменные. **Регистр в PHP имеет важное значение.**

**example\_1.** Немного о себе

```
<!doctype html>

<html lang="ru">

<head>

    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">

    <title>Программирование на языке PHP</title>

</head>

<body>

    <h1>Основы программирования</h1>

    <h2>Немного о себе</h2>

    <?php

        // запишем в переменную значение

        $color = "красного";

        echo "Мой феррари " . $color . " цвета<br>";

        echo "Мой коттедж " . $COLOR . " цвета<br>";

        echo "Моя яхта " . $coLoR . " цвета<br>";

        echo "<h2>Упс..., что-то в моей жизни пошло не так
        (</h2>";

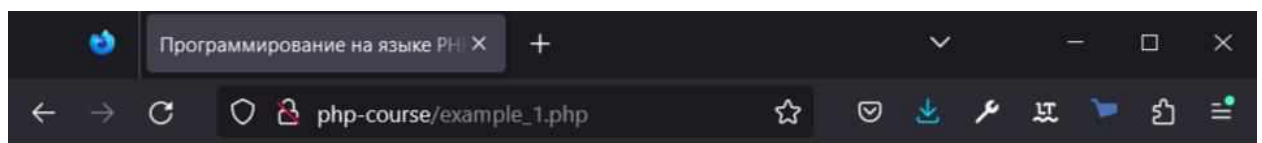
    ?>
```

```
</body>
```

```
</html>
```

В приведенном коде демонстрационного примера определена только переменная **\$color**, соответственно только первая инструкция вывода будет отображать значение переменной.

Использование в коде сценария необъявленных переменных вызывает ошибку типа **Warning**. Происходит это потому, что **\$color**, **\$COLOR**, и **\$coLOR** рассматриваются как три **различные переменные**.



## Основы программирования

### Немного о себе

Мой феррари красного цвета

**Warning:** Undefined variable \$COLOR in C:\OpenServer\domains\php-course\example\_1.php on line 19  
Мой коттедж цвета

**Warning:** Undefined variable \$coLOR in C:\OpenServer\domains\php-course\example\_1.php on line 20  
Моя яхта цвета

Упс..., что-то в моей жизни пошло не так (

**Примечание.** В приведенном примере для **сцепления** (соединения) строковых литералов и переменных используется операция **конкатенации**.

Переменная **\$color** — пример простого имени, но не все имена так просты. Довольно часто они **составные**, то есть включают в себя

несколько слов. Например, "имя пользователя", "дата рождения". В разных языках применяются разные стили именования.

В именовании переменных можно выделить несколько основных подходов, которые иногда комбинируют друг с другом. Все эти подходы проявляют себя, когда имя переменной состоит из **нескольких слов**:

- **kebab-case** — составные части переменной разделяются дефисом. Например: **my-super-var**.
- **snake\_case** — для разделения используется подчеркивание. Например: **my\_super\_var**.
- **CamelCase** — каждое слово в переменной пишется с заглавной буквы. Например: **MySuperVar**.
- **lowerCamelCase** — каждое слово в переменной пишется с заглавной буквы, кроме первого. Например: **mySuperVar**.

В PHP чаще всего стараются использовать **CamelCase** и его вариацию **lowerCamelCase**, при котором первая буква первого слова — строчная.

Впрочем, это всего лишь **рекомендации**.

## ***Выражения в определениях***

---

Переменные полезны не только для хранения и переиспользования информации, но и для **упрощения сложных вычислений**.

Рассмотрим пример: нужно перевести 50 евро в рубли через доллары. Подобные конвертации через промежуточную валюту часто делают банки при покупках за рубежом. Задачу выполним из расчета, что:

- отношение евро к **доллару**:  
 $1 \text{ EUR} = 1.1342 \text{ USD}$ ;
- отношение доллара к **рублю**:  
 $1 \text{ USD} = 74,71 \text{ RUB}$ .



Зная курсы валют, в коде примера **example\_2** произведем необходимые вычисления.

### **example\_2.** Выражения

```
<!doctype html>

<html lang="ru">

<head>

    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">

    <title>Программирование на языке PHP</title>

</head>

<body>

    <h1>Основы программирования</h1>

    <h2>Выражения</h2>

    <?php

        // переводим евро в доллары

        $dollarsCount = 50 * 1.1342; // => 56,71

        // переводим доллары в рубли

        $rublesCount = $dollarsCount * 74.71;

        echo "50 EUR = ";

        echo $rublesCount;

        echo " RUB";

    ?>
```

```
</body>
</html>
```

В предыдущих примерах мы записывали в переменную конкретное (простое) значение. Тогда как, здесь в:

```
$dollarsCount = 50 * 1.1342;
```

- справа от знака равно находится **выражение**. Интерпретатор вычислит результат — и запишет его в переменную **\$dollarsCount**.

**К сведению.** С точки зрения интерпретатора не важно, что перед ним: 56.71 или  $50 * 1.1342$ , для него **оба варианта — выражения**, которые надо вычислить. И они вычисляются в одно и то же значение — 56.71.

Правила построения кода (или грамматика языка) таковы, что в тех местах, где ожидается выражение, можно поставить **любое вычисление** (не только математическое), и программа останется работоспособной.

Любая переменная может быть частью любого выражения. В момент вычисления **вместо имени переменной подставляется её значение**.

Интерпретатор вычисляет значение **\$dollarsCount** до того, как эта переменная начнёт использоваться в других выражениях. Когда подходит момент использования переменной, РНР "знает" значение, потому что уже вычислил его.

## Магические числа

---

Вспомним пример из example\_2:

```
<?php
    // переводим евро в доллары
```

```
$dollarsCount = 50 * 1.1342  
  
// переводим доллары в рубли  
  
$rublesCount = $dollarsCount * 74.71;  
  
echo "50 EUR = ";  
  
echo $rublesCount;  
  
echo " RUB";
```

Такой код, не соответствует так называемым лучшим практикам профессиональной разработки. И причина здесь вот в чём: в программе фигурируют числа:

- 50
- 1.1342
- 74,71

Насколько высока вероятность, что через неделю вы вспомните что эти числа обозначают? А представьте, что будет через месяц!

А как его поймет новый программист, не видевший код ранее? В нашем примере **контекст восстанавливается** благодаря грамотному именованию, и оставленным комментариям, но в реальной жизни код значительно сложнее, и поэтому догадаться до смысла чисел зачастую невозможно.

**К сведению.** Магическими числами называются числа, происхождение которых невозможно понять без глубокого знания происходящего внутри данного участка кода (**контекст** кода).

Выход из ситуации прост: достаточно создать переменные с правильными именами, как всё встанет на свои места.

**example\_3.** Магические числа

```
<?php
```

```
// определения переменных

$dollarPerEuro = 1.1342;

$rublePerDollar = 74.71;

$eurosCount = 50;


// расчеты

$dollarsCount = $eurosCount * $dollarPerEuro;

$rublesCount = $dollarsCount * $rublePerDollar;


// вывод

echo "50 EUR = ";

echo $rublesCount;

echo " RUB";

?>
```

Обратите внимание на следующие детали:

- Именование переменных **lowerCamelCase**.
- Переменные отделены от последующих вычислений пустой строкой. Эти переменные имеют смысл и без вычислений, поэтому такое отделение уместно, оно повышает читаемость.
- Получился хорошо именованный и структурированный код, но он длиннее прошлой версии. Так часто бывает, и это нормально, потому что **код должен быть читабельным**.

## ***Ошибки при работе с переменными***

---

Переменная должна быть объявлена до её использования. Если сделать это позже, то программа просто не заработает.

```
<?php

print($message);
```

```
$message = 'Stop the World - I Want to Get Off!';
```

Запуск программы выше завершается с ошибкой PHP:

### **Warning: Undefined variable: \$message in ...**

— **это ошибка обращения**. Она означает, что в коде используется имя (говорят идентификатор), который не определен. Причём в самой ошибке явно указан идентификатор переменных: **\$message**.

**example\_4.** Порядок определения переменных важен

```
<?php

// вывод несуществующей переменной

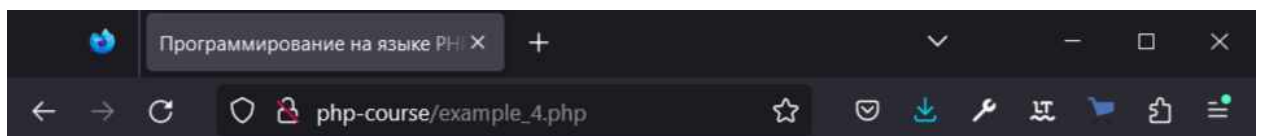
print($message);

// инициализация переменной строковым значением

$message = 'Stop the World - I Want to Get Off!';

?>
```

Собственно говоря, возникла ситуация, рассмотренная ранее – попытка использования несуществующей переменной.



## **Основы программирования**

### **Магические числа**

**Warning: Undefined variable \$message in C:\OpenServer\domains\php-course\example\_4.php on line 15**

Кроме неправильного порядка определения, в PHP встречаются банальные опечатки — как при использовании переменной, так и во время её объявления.

Количество подобных ошибок уменьшается за счёт использования правильно настроенного редактора. Такой редактор подсвечивает имена, которые используются без объявления, и предупреждает о возможных проблемах.

## Отчеты об ошибках

---

За уровень обработки ошибок в PHP отвечает **директива** (функция) **error\_reporting** конфигурационного файла интерпретатора PHP – **php.ini**. В PHP есть много уровней ошибок. Данная директива определяет типы ошибок, о которых PHP информирует выводом текстового сообщения в окно браузера.

**Важно.** Директива **error\_reporting** — задаёт, какие ошибки PHP попадут в отчёт.

### error\_reporting

**error\_reporting()** — устанавливает, какие ошибки PHP попадут в отчёт

#### Описание

```
error_reporting($error_level = null);
```

Функция **error\_reporting()** задаёт значение директивы **error\_reporting** во время выполнения программы. PHP содержит много уровней ошибок. Через эту функцию задают уровень ошибок на время работы (выполнения) скрипта, которые попадут в отчёт.

Если необязательный аргумент `error_level` не задан, функция **`error_reporting()`** вернёт текущее значение уровня протоколирования ошибок.

## Параметры

- **`error_level`** – новое значение уровня **`error_reporting`**. Значение параметра – именованная константа. При указании именованных констант нужно будет следить за совместимостью с новыми версиями PHP. По мере добавления уровней ошибок диапазон целых чисел увеличивается, поэтому старые уровни ошибок на основе целых чисел не всегда будут вести себя предсказуемо.

## Возвращаемые значения

Возвращает значение директивы **`error_reporting()`**, которое в ней хранилось до того, как оно было изменено на значение параметра **`error_level`**.

Замечание: Оператор управления ошибками (`@`) изменяет значение параметра `error_level` во время обработки ошибки.

В таблице приведены некоторые значения директивы **`error_reporting`**:

| Значение | Константа                     | Описание ошибки  |
|----------|-------------------------------|--|
| 1        | <b><code>E_ERROR</code></b>   | Фатальные ошибки времени выполнения. Это неустраняемые средствами самого скрипта ошибки, такие как ошибка распределения памяти и т.п. Выполнение скрипта в таком случае прекращается.            |
| 2        | <b><code>E_WARNING</code></b> | Предупреждения времени выполнения (не фатальные ошибки). Выполнение скрипта в таком случае не прекращается.  |
| 4        | <b><code>E_PARSE</code></b>   | Ошибки синтаксического анализатора (парсинга). Должны генерироваться только парсером.  |
| 8        | <b><code>E_NOTICE</code></b>  | Уведомления времени выполнения. Указывают на то, что во время выполнения скрипта произошло что-то, что может указывать на ошибку, хотя это может происходить и при обычном выполнении программы. |

|       |              |  |
|-------|--------------|--|
| 32767 | <b>E_ALL</b> | Все поддерживаемые ошибки, предупреждения и замечания. |
|-------|--------------|--|

\* - полный список констант можно посмотреть в открытых источниках

Ниже приведены некоторые примеры использования констант в директиве сценария.

```
// добавлять в отчёт все ошибки PHP

error_reporting(-1);

// добавлять в отчёт все ошибки PHP

error_reporting(E_ALL);

// включать в отчёт простые описания ошибок

error_reporting(E_ERROR | E_WARNING | E_PARSE);

// включать в отчёт E_NOTICE сообщения (добавятся сообщения о
непроинициализированных переменных или ошибках в именах
переменных)

error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// добавлять сообщения обо всех ошибках, кроме E_NOTICE

error_reporting(E_ALL & ~E_NOTICE);

// выключение протоколирования ошибок

error_reporting(0);
```

На основе демо примера **example\_3** поэкспериментируем с константами директивы **error\_reporting**.

#### **example\_5.** Константа E\_ERROR

```
<?php

// выводить только фатальные ошибки времени выполнения
```



```
error_reporting(E_ERROR);

// отсутствие переменной не фатальная ошибка
// $dollarPerEuro = 1.1342;

$rublePerDollar = 74.71;

$eurosCount = 50;

$dollarsCount = $eurosCount * $dollarPerEuro;

$rublesCount = $dollarsCount * $rublePerDollar;

echo "50 EUR = . $rublesCount;

echo " RUB";
```

?>

#### example\_6. Константа E\_PARSE

```
<?php

// выводить ошибки синтаксического анализатора

error_reporting(E_PARSE);

// переменная не определена, но это не ошибка парсинга

$dollarPerEuro;

$rublePerDollar = 74.71;

$eurosCount = 50;

$dollarsCount = $eurosCount * $dollarPerEuro;

$rublesCount = $dollarsCount * $rublePerDollar;

echo "50 EUR = " . $rublesCount;

echo " RUB";
```

?>

#### example\_7. Отключение отчета об ошибках

```
<?php
```

```
// отключить отчёт о всех ошибках

error_reporting(0);

$dollarPerEuro = 1.1342;
$rublePerDollar = 74.71;
$eurosCount = 50;
$dollarsCount = $eurosCount * $dollarPerEuro;
$rublesCount = $dollarsCount * $rublePerDollar;

// арифметический знак + для строки и числа - ошибка
// "50 EUR = " - строка
// $rublesCount - число

echo "50 EUR = " + $rublesCount;

echo " RUB";

?>
```

В открытых источниках можно увидеть аналогичную директиве **error\_reporting** функцию:

```
ini_set('error_reporting', E_ALL);
```

Обе формы записи функционально идентичны, за некоторыми незначительными на данный момент отличиями.

## ini\_set

**ini\_set()** — устанавливает значение настройки конфигурации.

### Описание

```
ini_set($option, $value);
```

Устанавливает значение заданной настройки конфигурации. Настройка будет хранить установленное значение пока выполняется скрипт. После завершения работы скрипта значение настройки вернётся к исходному.

## Параметры

- **option** – изменяемая настройка. Не все доступные настройки можно изменять функцией **ini\_set()**.
- **value** – новое значение настройки.

## Возвращаемые значения

В случае успешного выполнения возвращает **старое значение** или **false** в случае возникновения ошибки.

# Задание 2

=====

В директории раздаточного материала вам предложен файл **index.php**. Модернизируйте сценарий, в коде сценария создайте две пары PHP-тегов:

- В первой паре **сохраните** все фрагменты текста диалога в **строковые переменные**.
- Во второй паре **выведите** значения переменных **в браузер**.

```
<?php
```

```
    // инициализация переменных
```

```
?>
```

```
<?php
```

```
    // вывод переменных в браузер
```

```
?>
```

# Константы

- Синтаксис
- Магические константы
- Предопределенные константы

## Синтаксис

---

**Константа** - это идентификатор (имя) для простого значения. Как следует из названия, их значение не может измениться в ходе выполнения скрипта (кроме магических констант, которые на самом деле не являются константами).

Константы чувствительны к регистру. По принятому соглашению, имена констант всегда пишутся **в верхнем регистре**.

Имя константы должно соответствовать тем же правилам именования, что и другие имена в PHP. Правильное имя начинается с буквы или символа подчёркивания, за которым следует любое количество букв, цифр и символов подчёркивания. При этом **знак доллара не нужен**.

Константа может быть определена с помощью:

- ключевого слова **const**.
- функции **define()**.

**Важно.** После того, как константа определена, её значение не может быть изменено или аннулировано.

При использовании ключевого слова **const** допускаются только **скалярные** (простые) **выражения** и константы **array**, содержащие только скалярные выражения.

### example\_1. Ключевое слово const

```
<?php

    // скалярное значение

    const MYCONST = 'Земля прощай!';

    echo MYCONST;

    echo "<br />";

    // скалярное выражение

    const ANOTHER_CONST = MYCONST . ' В добрый путь!';

    echo ANOTHER_CONST;

    echo "<br />";

    // массив скалярных значений

    const AUTHOR = array('Довлатов', 'Чехов', 'Ерофеев');

    echo AUTHOR[0]; // Довлатов

    echo AUTHOR[1]; // Чехов

    echo AUTHOR[2]; // Ерофеев

?>
```

Функция **define()** позволяет задать константу через **выражение**.

### example\_2. Функция define

```
<?php

    // значение константы - строка

    define("MYCONST", "Здравствуй, мир!");

    echo MYCONST;

    echo "<p>";

    // значение константы - массив

    define('WEB', array(

        'PYTHON',
```

```

        'PHP',

        'PERL'

    ));

    echo WEB[1];

    echo "<p>";

    // значение константы - выражение

    define("RADIUS", 3.14 * 9);

    echo RADIUS;

?>

```

В следующем примере в названии константы *случайно* затесался русский символ (С).

### **example\_3.** Ошибка в именовании константы

```

<?php

    // отключаем протоколирование ошибок

    error_reporting(0);

    // определяем константу MYCITATE

    define("MYCITATE", "Здравствуй, мир!");

    // пытаемся вывести значение константы в браузер

    /*

        MYCITATE <> MYC(Ru) ITATE

    */

    echo "<h2>" . MYCITATE . "</h2>";

?>

```

Различия между константами и переменными:

- у констант нет приставки в виде знака доллара (\$);

- константы могут быть определены и доступны **в любом месте без учёта области видимости** (об этом позже);
- константы **не могут быть переопределены или удалены** после первоначального объявления;
- константы могут иметь только скалярные значения, выражения или массивы.

## ***Магические константы***

---

Кроме обычных констант, в PHP существует отдельная группа — **магические константы**. Их отличия заключаются в следующем:

- магические константы невозможно определить самому, можно пользоваться только **существующими**;
- магические константы **начинаются и заканчиваются символами \_\_** (два подчёркивания);
- магия заключается в том, что такие константы имеют одно и то же значение только в пределах **определенной части** программы.

Последний пункт может показаться странным. Какие же они константы, если их значение меняется? Скажем так, они не очень постоянные, но их изменения четко регламентированы и на практике не вызывают проблем.

**Важно. Магическая константа** - специальная константа, доступная в PHP. Содержит разное значение в разных контекстах.

К подобным константам относятся, например, такие:

- **\_\_LINE\_\_** — содержит текущую строку файла, в котором она используется.
- **\_\_FILE\_\_** — путь до текущего файла.
- **\_\_DIR\_\_** — путь до директории, в которой находится текущий файл.



**Есть девять магических констант**, которые меняют своё значение **в зависимости от контекста**, в котором они используются. Например, значение `__LINE__` зависит от строки в скрипте, на которой эта константа указана.

Все магические константы определяются во время **компиляции**, в отличие от обычных констант, которые определяются во время **выполнения**.

Магические константы **нечувствительны** к регистру.

**example\_4.** Некоторые магические константы

```
<?php

    echo "Директория, в которой находится текущий файл: <b>";
    echo __DIR__;
    echo "</b><p>";

    // магическая константа нечувствительна к регистру

    echo "Путь до текущего файла: <b>";
    echo __file__;
    echo "</b><p>";

    echo "Текущая строка исполняемого файла: <b>";
    echo __LINE__;
    echo "</b>";

?>
```

Нечувствительность к регистру дает право именовать магические константы так, как вам нравится.

# Предопределенные константы

---

PHP предоставляет большой список **предопределённых констант** для каждого выполняемого скрипта.

Вещь достаточно специфическая, и в моем проекте не используется, но знать о существовании надо.

Многие из этих констант определяются различными модулями и будут присутствовать только в том случае, если эти модули доступны в результате динамической загрузки или в результате статической сборки.

Ниже представлены некоторые константы из списка предопределенных.

## example\_5. Предопределенные константы

```
<?php

echo "Текущая версия PHP: <b>";

echo PHP_VERSION;

echo "</b><p>";

echo "Операционная система, под которую собирался PHP: <b>";

echo PHP_OS;

echo "</b><p>";

echo "Максимальная длина файловых имён (включая путь),  
поддерживаемая данной сборкой PHP: <b>";

echo PHP_MAXPATHLEN;

echo "</b><p>";

echo "Максимальное целое число, поддерживаемое данной сборкой  
PHP: <b>";

echo PHP_INT_MAX;

echo "</b><p>";

echo "Максимальное возможное число типа float: <b>";
```

```
echo PHP_FLOAT_MAX;
```

```
echo "</b><p>";
```

```
?>
```

**Из опыта.** В подавляющем количестве случаев опыт использования констант сводится к применению **магических констант** и констант, определяемых параметрами **подключения к базе данных**.

# Задание 3

=====

Используйте решение файла **index.php** раздаточного материала.

Создайте **константу**, содержащую в качестве значения **массив из трех элементов**:

- 'М.А. Булгаков'
- 'Мастер и Маргарита'
- '1966'

**Под основным текстом** страницы выведите значения созданной константы. При выводе используйте HTML-тег авторского форматирования (`<pre>...</pre>`). Формат вывода может быть, например, такой:

Автор: [автор]

Произведение: [произведение]

Год издания: [год\_издания]

**P.S.** На место шаблона [...] подставляются соответствующие значения массива константы.

# Типы данных

- Введение
- Слабая типизация
- Типы данных языка PHP
- Явное преобразование типов

## Введение

---

Бывают разные способы представлять данные в программах.

Есть **строки** — наборы символов в кавычках, вроде "Hello, World!". Есть **целые числа** — например, 7, -198, 0. Это две разные категории информации — два разных **типа данных**.

Операция умножения имеет смысл для целых чисел, но не имеет смысла для строк: умножать слово "php" на слово "программирование" — бессмыслица.

**Важно.** Тип данных определяет, что можно делать с элементами конкретного множества элементов. Строки, целые числа, рациональные числа — это **разные типы данных**.

Язык программирования распознает типы. Поэтому PHP не позволит нам умножать строку на строку ("умножать текст на текст"), но позволит умножать целое число на другое целое число. Наличие типов и таких ограничений в языке защищает программы от случайных ошибок.

В отличие от строк, числа оборачивать в кавычки не нужно. Чтобы напечатать число 5, достаточно написать:

```
<?php  
echo 5;
```

Обратите внимание, что число 5 и строка '5' — совершенно разные вещи, хотя вывод через print для этих данных идентичный.

Целые числа (1, 34, -19 и т.д.) и вещественные числа (1.3, 1.0, -14.324 и т.д.) — это два **отдельных типа** данных.

Вот ещё один пример, но уже с вещественным числом:

```
<?php
    print(10.234);
```

Типы данных **строка, целое число, вещественное число** — это **примитивные типы**, они встроены в сам язык PHP. В язык встроены также и некоторые **смешанные типы** данных, но пока мы будем работать только с примитивными. Программисты также могут создавать **собственные типы данных**.

## Слабая типизация

---

Нам известно про два разных типа данных: числа и строки. Мы, например, могли складывать числа, потому что операция сложения — это операция для типа "числа".

А что, если применить эту операцию не к двум числам, а к числу и строке?

**example\_1.** Слабая типизация

```
<?php
    /*
        в слабо типизированных языках
        сложить число и строку...
        почему бы и нет ?
    */
```

```
print(1 + '7');
```

```
?>
```

Несмотря на то, что '7' — это строка, а не число, интерпретатор PHP выдал ответ 8, как если бы мы складывали два числа.

Когда PHP видит несоответствие типов, он сам **пытается преобразовать** данные. В данном случае он преобразовал строку '7' в число 7, а потом спокойно сложил 1 и 7.

Не все языки так делают.

**Важно.** PHP — это язык со **слабой типизацией**.

PHP знает о существовании разных типов (числа, строки и др.), но относится к их использованию не очень строго, пытаясь преобразовывать данные, когда ему это кажется разумным.

В языках со **строгой типизацией** сложить число со строкой не получится.

PHP был создан для глобальной сети Интернет, а в интернете вся информация — это строки. Даже когда вы вводите на сайте номер телефона или год рождения, на сервер эта информация поступает не как число, а как строка. Поэтому авторы языка решили, что автоматически преобразовывать типы — правильно и удобно.

Такое автоматическое **неявное преобразование** типов с одной стороны удобно. Но на практике это свойство языка создает множество ошибок и проблем, которые трудно найти. Код может иногда работать, а иногда не работать — в зависимости от того, "повезло" ли в конкретном случае с автоматическим преобразованием. Программист это заметит не сразу.

Слабая типизация красной нитью проходит сквозь всю разработку на PHP.

# Типы данных языка PHP

---

PHP поддерживает следующие типы данных:

- **Скалярные** типы данных
  - Строка (String)
  - Целое число (Integer)
  - Вещественное число (Float)
  - Булевы (Boolean)
- **Смешанные** типы данных
  - Массив (Array)
  - Объект (Object)
- **Специальные** типы данных
  - Пустой тип (Null)
  - Ресурс (Resource)

**Еще раз. Скалярный** или говорят еще простой **тип** данных включает в себя **булев, числовые, строковые** данные, а также **ссылки**.

Рассмотрим некоторые типы данных. Для анализа некоторых типов (особенно смешанных) функция **echo** не лучший вариант, совсем.

**example\_2.** Функция echo и массив

```
<?php

// выводить только критичные ошибки
error_reporting(E_ERROR);

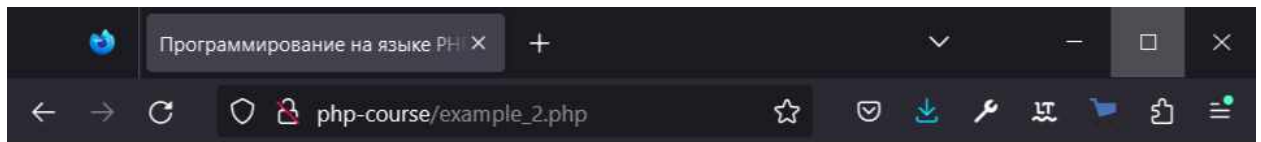
// вот такой вот массив
$arr = ["php", "селедка", "шпрехшталмейстер"];

// вывод массива через функцию echo - не работает

echo $arr;

?>
```





# Основы программирования

## Типы данных

Array

Дизайн | Верстка | Веб-программирование

При рассмотрении типов удобно использовать функции **var\_dump()** или **print\_r()**. В дальнейшем мы всегда будем использовать данные функции. Особенно при работе со сложными структурами данных.

### *К сведению.*

- **var\_dump()** — выводит структурированную информацию о переменной, включая ее тип и значение.
- **print\_r()** — выводит структурированную информацию о переменной в удобочитаемом виде.

## Строка (String)

**Строка** - последовательность символов.

Строкой может быть **любой текст** внутри кавычек. Вы можете использовать **одинарные** или **двойные** кавычки. Разница между кавычками весьма существенная, но об этом немного позже. Не надо использовать вложенные кавычки (по крайней мере до тех пор, пока мы не разберем особенность вложения).

### example\_3. Тип данных Строка

```
<h1>Основы программирования</h1>

<h2>Тип данных</h2>

<hr>

<h2>Тип данных - Строка</h2>

<?php

    var_dump ( 'Привет Мир! ' );

?>

</body>
</html>
```

## Целое число (Integer)

**Целочисленный** тип данных - целое десятичное число.

Правила для целых чисел:

- целое число должно иметь по крайней мере одну цифру;
- целое число не должно иметь запятой;
- целое число может быть положительным или отрицательным;
- целые числа могут быть указаны в трех форматах: десятичным (10-разрядный), шестнадцатеричным (16-разрядный - с префиксом 0x) или восьмеричным (8-разрядный - с префиксом 0).

#### example\_4. Тип данных Целое число

```
<h1>Основы программирования</h1>

<h2>Типы данных</h2>

<hr>

<h2>Тип данных - Целое число</h2>

<?php
    var_dump(250);
?>

</body>
</html>
```

## Вещественное число (Float)

**Вещественный** тип данных - десятичное число с плавающей точкой.

#### example\_5. Тип данных Вещественное число

```
<h1>Основы программирования</h1>

<h2>Типы данных</h2>

<hr>

<h2>Тип данных - Число с плавающей точкой</h2>

<?php
```

```
var_dump(25.10);

?>

</body>
</html>
```

## Булев (Boolean)

**Логический** тип данных отражает два возможных состояния: **true** или **false**.

Логические операторы часто используются в условных конструкциях.

**example\_6.** Тип данных Булев

```
<h1>Основы программирования</h1>

<h2>Типы данных</h2>

<hr>

<h2>Тип данных - Булев</h2>

<?php

    var_dump(true);

?>
```

```
</body>
```

```
</html>
```

## Массив (Array)

Тип данных **Массив** хранит несколько значений в одной переменной. Массивам посвящена целая тема конспекта. Массивы анализируются рекурсивно с разным отступом у значений для визуального отображения структуры.

**example\_7.** Тип данных Массив

```
<h1>Основы программирования</h1>
```

```
<h2>Типы данных</h2>
```

```
<hr>
```

```
<h2>Тип данных - Массив</h2>
```

```
<?php
```

```
    echo "<pre>";
```

```
    var_dump(array("РНР", "Пассатижи", "Балалайка"));
```

```
    echo "</pre>";
```

```
    echo "<pre>";
```

```
    print_r(array("РНР", "Пассатижи", "Балалайка"));
```

```
    echo "</pre>";
```

```
?>
```

```
</body>
```

```
</html>
```

## Пустой тип (Null)

**Пустой** тип - специальный тип данных, который может иметь только одно значение: **Null**.

Если переменная создается без значения, то ей **автоматически** **присваивается** значение Null.

**example\_8.** Тип данных Null

```
<h1>Основы программирования</h1>
```

```
<h2>Типы данных</h2>
```

```
<hr>
```

```
<h2>Тип данных Null</h2>
```

```
<?php
```

```
    var_dump(null);
```

```
?>
```

```
</body>
```

```
</html>
```

# Явное преобразование типов

---

Преобразование типов довольно частая операция в веб-разработке. С одной стороны, для её выполнения можно **полагаться на слабую типизацию**, с другой — во многих ситуациях лучше делать преобразование явно, используя специальный синтаксис. **Такой код понятнее и более предсказуем.**

**example\_9.** Явное преобразование типов

```
<h1>Основы программирования</h1>

<h2>Типы данных</h2>

<hr>

<h2>Явное преобразование типов</h2>

<?php

    print((string) 5);

    print('<br />');

    print((int) '345');

?>

</body>

</html>
```

**Примечание.** При явном **преобразовании типа** - перед

преобразуемым значением, в скобках, **указывается желаемый тип**.

В результате преобразования значение справа преобразуется в значение другого типа, указанного слева в скобках. Преобразование работает для любого типа данных.

Преобразование типов можно использовать **внутри составных выражений**. Дополнительные скобки помогают визуально отделить части выражения друг от друга:

```
print('Это ' . ((string) 5));
```

В более сложных ситуациях встречаются **множественные преобразования типов**:

```
(string) (5 + ((int) '4'));
```

Порядок вычисления этого выражения следующий:

1. (int) '4'; => 4
2. 5 + 4; => 9
3. (string) 9; => '9'

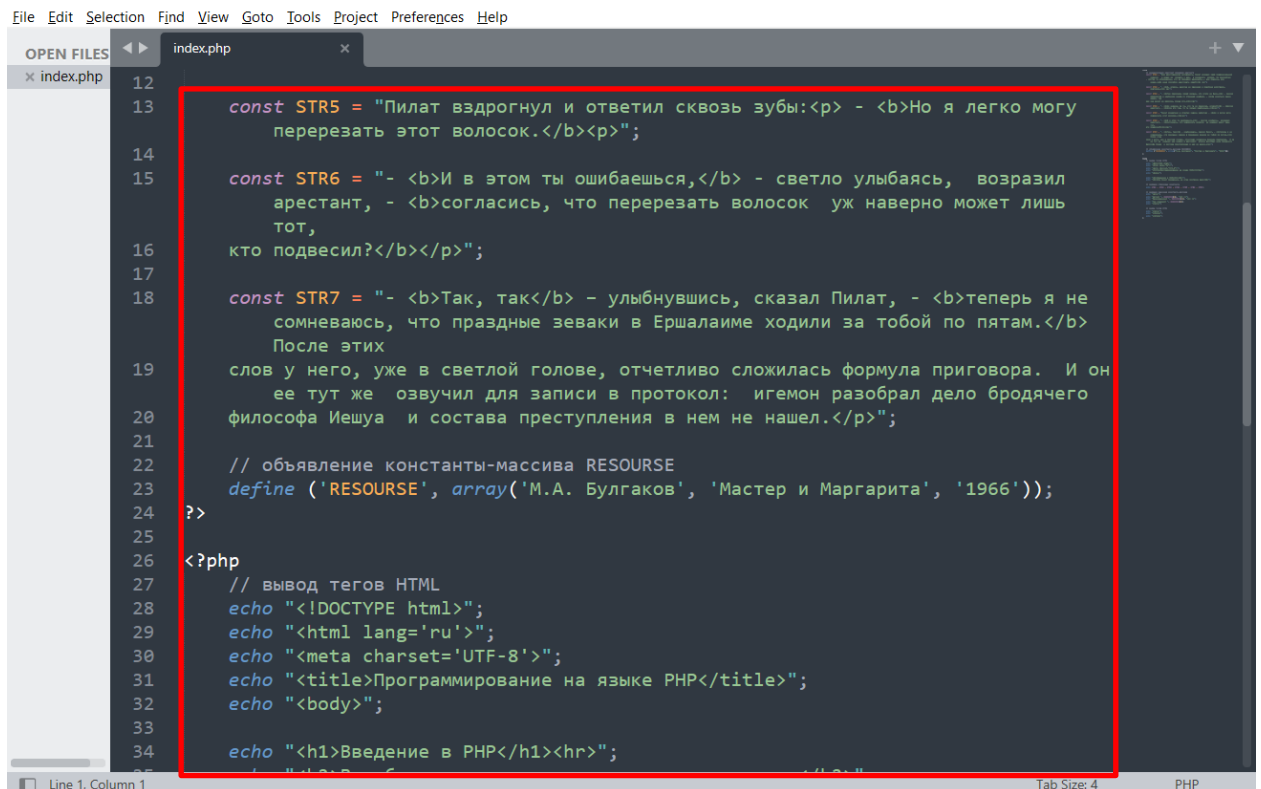


# Задание 4

=====

В директории раздаточного материала вам предложен файл **index.php**.

Выполните вывод содержимого страницы с помощью **PHP-инструкций echo** или **print** (все содержимое, включая HTML-код, выводится через PHP инструкцию echo / print).



```
12
13  const STR5 = "Пилат вздрогнул и ответил сквозь зубы:<br> - <b>Но я легко могу
14  перерезать этот волосок.</b><br>";
15
16  const STR6 = "- <b>И в этом ты ошибаешься,</b> - светло улыбаясь, возразил
17  арестант, - <b>согласись, что перерезать волосок уж наверно может лишь
18  тот,
19  кто подвесил?</b></p>";
20
21  const STR7 = "- <b>Так, так</b> - улынувшись, сказал Пилат, - <b>теперь я не
22  сомневаюсь, что праздные зеваки в Ершалаиме ходили за тобой по пятам.</b>
23  После этих
24  слов у него, уже в светлой голове, отчетливо сложилась формула приговора. И он
25  ее тут же озвучил для записи в протокол: игемон разобрал дело бродячего
26  философа Иешуа и состава преступления в нем не нашел.</p>";
27
28  // объявление константы-массива RESOURCE
29  define ('RESOURCE', array('М.А. Булгаков', 'Мастер и Маргарита', '1966'));
30
31  ?>
32
33  <?php
34  // вывод тегов HTML
35  echo "<!DOCTYPE html>";
36  echo "<html lang='ru'>";
37  echo "<meta charset='UTF-8'>";
38  echo "<title>Программирование на языке PHP</title>";
39  echo "<body>";
40
41  echo "<h1>Введение в PHP</h1><hr>";
```

# Задание 5

=====

Пусть имеется **высказывание**: "Тип данных зависит от способа представления данных в программе". Таким образом, тип данных значения будет зависеть от переменной, в которой хранится значение в программе.

```
<?php
```

```
$VarStr = 'Слабая типизация PHP';  
  
const CONSTSTR = 'Слабая типизация PHP';  
  
define("ARRSTR", array('Слабая типизация PHP'));
```

Напишите сценарий, используя конструкцию **var\_dump()** проверьте **истинность** или **ложность** представленного высказывания.

# Строки

- Одинарные кавычки
- Двойные кавычки
- Heredoc синтаксис
- Nowdoc синтаксис
- Объединение строк

## Одинарные кавычки

---

В веб-разработке программы постоянно оперируют строками. Все, что мы видим на сайтах, так или иначе представлено в виде текста.

Строка может быть определена четырьмя различными способами:

1. **Одинарными** кавычками.
2. **Двойными** кавычками.
3. **Heredoc**-синтаксисом.
4. **Nowdoc**-синтаксисом.

Простейший способ определить строку — это заключить ее в **одинарные кавычки** (').

Чтобы использовать **одинарную кавычку** внутри строки, определенной с помощью одинарных кавычек, проэкранируйте её обратным слешем (\).

```
<?php
    // так не работает
    echo 'Д'артаньян - супергерой моего детства';

    // а вот так работает
    echo 'В детстве я хотел стать Д\'артаньяном';
```

Если необходимо написать сам **обратный слеш**, продублируйте его (\\).

Все остальные случаи применения обратного следа будут интерпретированы как **обычные символы**: это означает, что, если вы попытаетесь использовать другие управляющие последовательности, такие как `\r` или `\n`, они будут выведены, как есть, вместо какого-либо особого поведения (об управляющих последовательностях смотри ниже).

**Важно.** Переменные и управляющие последовательности специальных символов в строках с одинарными кавычками, не обрабатываются.

Это означает, что переменная внутри строки одинарных кавычек не будет выводить свое значение, а будет отображаться как есть.

Приведу пример использования одинарных кавычек. При выполнении анализа кода скрипта используйте просмотр страницы в режиме разработчика – клавиша **F12**, вкладка **Elements** (название вкладки может меняться, в зависимости от браузера).

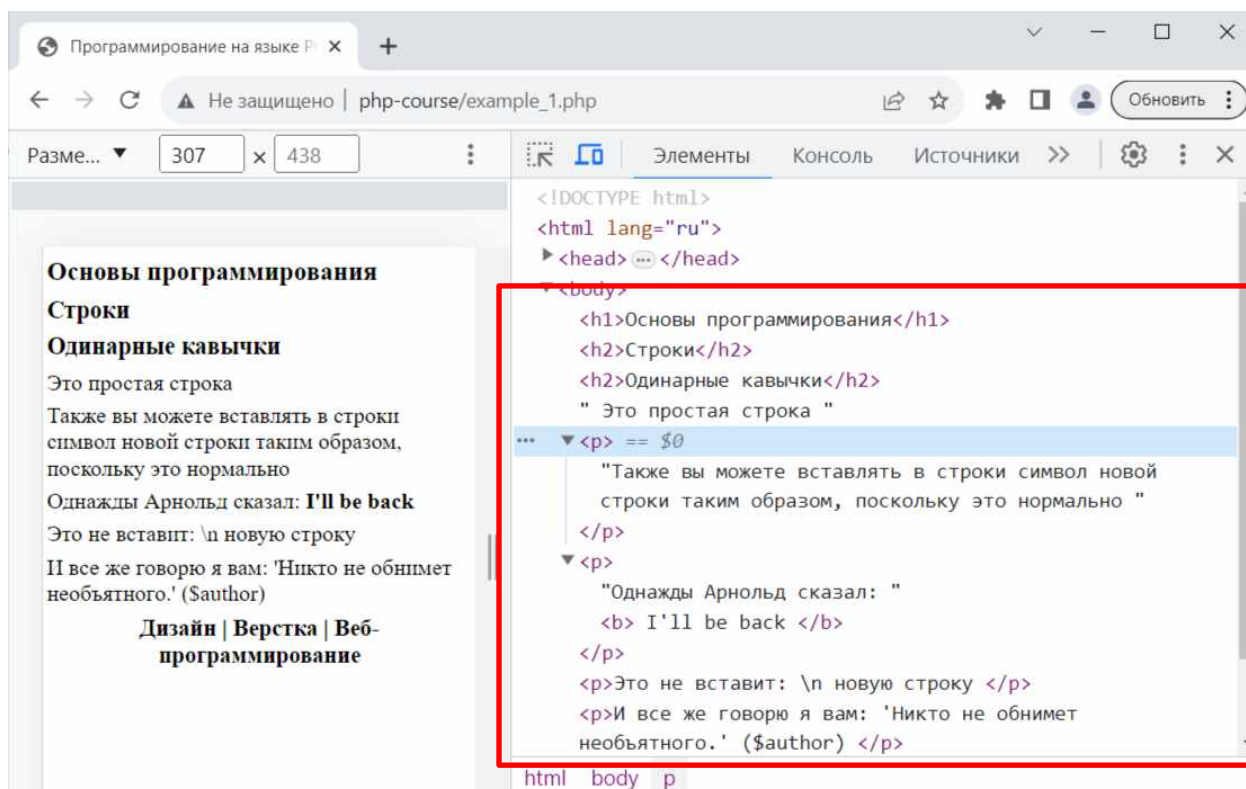


Рис.1. Анализ кода серверного скрипта

## example\_1. Использование одинарных кавычек

```
<?php
```

```
echo 'Это простая строка <p>';
```

```
/*
```

```
браузер все выведет в одну строку, потому что:
```

```
-- браузер не понимает всех ваших переводов строки в редакторе кода
```

```
-- браузер понимает HTML-теги
```

```
*/
```

```
echo 'Также вы можете вставлять в строки
```

```
символ новой строки таким образом,
```

```
поскольку это нормально <p>';
```

```
// в одинарных кавычках - апостроф. Браузер подумает, что это конец строки, а чтобы он так не подумал - экранируем апостроф обратным слешем
```

```
echo 'Однажды Арнольд сказал: <b> I\'ll be back </b><p>';
```

```
// в одиночных кавычках управляющие последовательности не работают (\n - перевод на новую строку)
```

```
echo 'Это не вставит: \n новую строку <p>';
```

```
$author = "Козьма Прутков";
```

```
// переменные внутри одинарных кавычек не подставляются
```

```
echo 'И все же говорю я вам: \'Никто не обнимет необъятного.\' ($author)';
```

```
?>
```

Внимательно изучайте предложенный код и сопровождающие его комментарии, не забывайте экспериментировать.

## Двойные кавычки

---

Другой способ определить строку - заключить символы в **двойные кавычки** (" ").

Если строка заключена в двойные кавычки ("), PHP **распознает управляющие последовательности специальных символов**.

Но самым важным свойством строк в двойных кавычках является **обработка переменных**.

**Важно.** Если строка указывается в **двойных кавычках**, либо при помощи **HEREDOC**-синтаксиса, переменные внутри неё **обрабатываются** (выводятся).

Обрабатываться могут не только переменные, но и значения **массива** (array) или свойства **объекта** (object). Такая возможность подстановки переменных в строку называется **интерполяцией**.

**example\_2.** Использование двойных кавычек

```
<?php

echo "Это простая строка <p>";

/*
браузер все выведет в одну строку, потому что:
-- браузер не понимает всех ваших переводов строки в
редакторе кода
-- браузер понимает HTML-теги
*/

echo "Также вы можете вставлять в строки
символ новой строки таким образом,
поскольку это нормально <p>";
```

```
// одинарные кавычки внутри двойных - экранировать не надо,
// браузер и без этого понимает, где конец строки

echo "Однажды Арнольд сказал: <b> I'll be back </b><p>";

// в двойных кавычках управляющие последовательности
работают (\n - перевод на новую строку)

echo "Это вставит: \n новую строку <p>";

$author = "Козьма Прутков";

// переменные внутри двойных кавычек подставляются
// экранировать одинарные кавычки не нужно

echo "И все же говорю я вам: 'Никто не обнимет необъятного.'
($author) ";

?>
```

## Немного об управляющих последовательностях

**Управляющая последовательность** (экранированная последовательность, от англ. escape sequence) — совокупность идущих подряд значащих элементов, **в группе** теряющих для обрабатывающего механизма своё **индивидуальное значение** с одновременным **приобретением этой группой нового значения**.

Некоторые из управляющих последовательностей представлены в таблице.

| Последовательность | Значение                 |
|--------------------|--------------------------|
| \n                 | Новая строка             |
| \r                 | Возврат каретки          |
| \t                 | Горизонтальная табуляция |
| \v                 | Вертикальная табуляция   |
| \\                 | Обратная косая черта     |

|     |                 |
|-----|-----------------|
| \\$ | Знак доллара    |
| \"  | Двойная кавычка |

Для такого случая я создал директорию **example\_3**, в которой созданы файлы со сценариями, демонстрирующими назначение некоторых **управляющих последовательностей**.

**Важно.** Файлы сценариев созданы не для изучения находящегося в них кода (для этого будет отдельная тема). Файлы, созданы для анализа **записываемых в текстовые файлы строк**.

После запуска каждого из скриптов анализируйте содержимое соответствующего текстового файла.

## **HEREDOC синтаксис**

---

Третий способ определения строк — это использование **HEREDOC**-синтаксиса: <<<.

После оператора Heredoc необходимо указать **идентификатор**, затем перевод строки. После этого идёт сама строка, а потом этот же **идентификатор, закрывающий вставку**.

**Важно.** После **открывающего идентификатора** не должно быть никаких символов, включая символы пробела – сразу **перевод строки**.

Базовый синтаксис **HEREDOC-строк**:

// базовый пример использования HEREDOC-синтаксиса



```
echo <<<HERE
```

```
    строка 1
```

```
    строка 2
```

```
    строка 3
```

```
    . . . .
```

```
HERE;
```

**Имя идентификатора можете придумать свое**, но имя должно соответствовать тем же правилам именования, что и любая другая метка в PHP: оно должно содержать только буквенно-цифровые символы и подчёркивания и должно начинаться с нецифрового символа или символа подчёркивания.

```
// имя идентификатора можно придумать свое
```

```
echo <<<DOC
```

```
    . . . .
```

```
DOC;
```

До PHP 7.3.0 закрывающий идентификатор должен был находиться в **самом начале новой строки**. В более новых версиях закрывающий идентификатор может иметь отступ с помощью пробела или табуляции.

**Примечание.** Однако необходимо учитывать следующий факт, если **закрывающий** идентификатор смещён дальше, чем любая строка тела, возникнет ошибка.

В HEREDOC синтаксисе строк нет ничего сложного, но с идентификатором необходимо обращаться аккуратно.

```
// так можно
```

```
$html = <<<DOC
```

```
    Имя: $name <br>
```

```
    Фамилия: $surname <br>
```

```
DOC;  
  
//  
// а так нельзя  
  
$html = <<<STR  
  
Имя: $name <br>  
  
Фамилия: $surname <br>  
  
STR;
```

#### example\_4. Расположение идентификатора

```
<?php  
  
$name = "Остап";  
  
$surname = "Бендер-Задунайский";  
  
// так можно  
  
$html = <<<DOC  
  
Имя: $name <br>  
  
Фамилия: $surname <br>  
  
DOC;  
  
echo $html;  
  
echo "<hr>";  
  
// а так нельзя  
  
// STR - не должен выступать  
  
$html = <<<STR  
  
Имя: $name <br>  
  
Фамилия: $surname <br>  
  
STR;  
  
echo $html;  
  
?>
```

Строки, объявленные с помощью HEREDOC-синтаксиса, поддерживают **интерполяцию** переменных.

#### **example\_5.** HEREDOC синтаксис

```
<?php

    // определим переменные

    $city = "Рим";

    $car = "Range Rover";

    $book = "Мастер и Маргарита";

    $team = "Pink Floyd";

    $symbol = "$";

    // <<< - HEREDOC-синтаксис

    // HERE - идентификатор, имя можете использовать любое

    echo <<<HERE

        Мой любимый город - $city. <br/>

        Моя любимая машина - $car. <br/>

        Мое любимое произведение - $book. <br/>

        Моя любимая группа - $team. <br/>

        Мой любимый символ - $symbol

    <p>

    HERE;

    // можно использовать так

    $html = <<<DOC

        Мой любимый город - $city. <br/>

        Моя любимая машина - $car. <br/>

        Мое любимое произведение - $book. <br/>

        Моя любимая группа - $team. <br/>

        Мой любимый символ - $symbol
```

```
DOC;  
  
echo $html;  
  
/*  
  
ВНИМАНИЕ!  
  
Закрывающий идентификатор в версии PHP до 7.3 должен быть  
прижат к левой стороне.  
  
В более новых версиях PHP это не обязательно, НО  
двигать его вправо за границу текста НЕЛЬЗЯ, ИНАЧЕ ОШИБКА  
  
*/  
  
?>
```

Если интерпретатор встречается в строке знак доллара (\$), он захватывает так много символов, сколько возможно, чтобы сформировать **правильное имя переменной**. Если вы хотите точно определить конец имени, заключайте имя переменной в фигурные скобки (**{}**).

#### **example\_6.** Использование фигурных скобок

```
<?php  
  
$juice = "apple";  
  
echo "He drank some $juice juice.<p>";  
  
// некорректно, 's' - верный символ для имени переменной, но  
переменная имеет имя $juice но не $juices  
  
echo "He drank some juice made of $juices.<p>";  
  
// корректно, строго указан конец имени переменной с помощью  
фигурных скобок  
  
echo "He drank some juice made of ${juice}s.<p>";  
  
?>
```

**К сведению.** При использовании **сложных переменных**

(многомерных массивов, объектов) **без фигурных скобок не обойтись.**

## ***NOWDOC синтаксис***

**NOWDOC** — это то же самое для строк в одинарных кавычках, что и **HEREDOC** для строк в двойных кавычках.

**К сведению.** NOWDOC похож на HEREDOC, но внутри него **не осуществляется никаких подстановок.**

NOWDOC указывается той же последовательностью <<<, что используется в HEREDOC, но последующий за ней идентификатор **заключается в одинарные кавычки**, например, <<<'EOT' (только отрывающий).

Все условия, действующие для идентификаторов HEREDOC также действительны и для NOWDOC, особенно те, что относятся к закрывающему идентификатору.

### **example\_7.** NOWDOC синтаксис

```
<?php

    // определим переменные

    $city = "Рим";

    $car = "Range Rover";

    $book = "Мастер и Маргарита";

    $team = "Pink Floyd";

    $symbol = "$";

    echo <<<'HERE '

    <h3>В программе мы определили следующие
```

```
переменные:</h3>

$city = "Рим" <br />

$car = "Range Rover" <br />

$book = "Мастер и Маргарита" <br />

$team = "Pink Floyd" <br />

$symbol = "$";

HERE;

?>
```

## Объединение строк

---

Веб-разработка чаще всего предполагает динамическое создание текстовых блоков, полученных из разных источников, которые затем объединяются вместе.

Объединение строк, возможно несколькими способами:

- **Конкатенация** (склеивание, сцепка) строк.
- **Интерполяция** (встраивание переменных в строку).
- Функции работы со строками и оператор **конкатенационного присваивания** ".="

**Конкатенация** (склеивание) строк всегда происходит в том же порядке, в котором записаны операнды. Левый операнд становится левой частью строки, а правый — правой.

### example\_8. Конкатенация

```
<?php

// объединяем строки конкатенацией

echo

"Всю жизнь я дул в подзорную трубу и удивлялся, что нету
```

```

музыки." .

" А потом внимательно глядел в тромбон и удивлялся, что ни
хрена не видно" .

" (С.Довлатов) ";

echo "<hr>";

$str1 = "Мне стало противно, и я ушёл.";
$str2 = " Вернее остался";

echo $str1 . $str2 . " (С.Довлатов) .<p>";

echo "<hr>";

// просто так ))

echo 'Порядочный человек – это тот, кто делает гадости без
удовольствия (С.Довлатов) .';

?>

```

**К сведению. Пробел** — такой же символ, как и другие, поэтому сколько пробелов поставить в строке — столько и получится.

В демонстрационном примере **example\_9** рассмотрим разные способы объединения строк.

#### **example\_9.** Объединение строк

```

<?php

// определяем набор переменных

$album = "The Dark Side of The Moon";

$team = "Pink Floyd";

$data = "17 марта 1973";

$label = "Harvest, Capitol, EMI";

$format = "LP, кассета, CD, SACD";

```

```

$status = "Платиновый (USA), Платиновый(GBR)";

$href = "https://ru.wikipedia.org/wiki/Pink_Floyd";

$img = "The_Dark_Side_of_The_Moon.jpg";

// синтаксис HEREDOC

echo <<<STR

<h2>$album</h2>

<b>Дата выхода:</b> $data<br>

<b>Группа:</b> <a href='$href' target='blank'> $team
</a><br>

<b>Лейбл:</b> $label <br>

<b>Формат:</b> $format <br>

<b>Статус:</b> $status <p>

<img src='$img'>

STR;

echo "<hr>";

// синтаксис одинарных кавычек и конкатенации

echo '<h2>' . $album . '</h2>' .

'<b>Дата выхода:</b>' . $data . '<br>' .

'<b>Группа:</b> <a href=' . $href . ' target=\'blank\'>' .
$team . '</a><br>' .

'<b>Лейбл:</b>' . $label . '<br>' .

'<b>Формат:</b>' . $format . '<br>' .

'<b>Статус:</b>' . $status . '<p>' .

'<img src=' . $img . '>';

echo "<hr>";

// синтаксис двойных кавычек и интерполяции

echo "

```



```
<h2>$album</h2>

<b>Дата выхода:</b> $data<br>

<b>Группа:</b> <a href='$href' target='blank'> $team
</a><br>

<b>Лейбл:</b> $label <br>

<b>Формат:</b> $format <br>

<b>Статус:</b> $status <p>

<img src='$img'>

";
```

?>

# Задание 6

=====

В файле **диалог.txt** уже знакомый вам диалог Понтия Пилата и Иешуа. Текст разбит на семь фрагментов. Прямая речь заключена в одинарные кавычки.

С помощью любой из инструкций, используя **двойные кавычки**, выведите диалог в браузер.

**Отформатируйте** вывод.

# Задание 7

=====

Оформите вывод фрагмента текста (шаблона) с использованием синтаксиса **HEREDOC**.

Используя **интерполяцию**, в места подстановки шаблона вставьте соответствующие данные.

-----

Дав арестованному договорить, Пилат изменил свой первоначальный замысел и решил не спорить с ним, а закончить допрос. Он сказал:<p>

– <b>[hegemon1]</b><p>

– <b>[yesua1]</b><p>

– <b>[hegemon2]</b> – сказал прокуратор и улыбнулся какой-то страшной улыбкой. – <b>[hegemon3]</b></p>

– <b>[yesua2]</b> – спросил арестант. – <b>[yesua3]</b><p>

Пилат вздрогнул и ответил сквозь зубы:<p> – <b>[hegemon4]</b><p>

– <b>[yesua4]</b> – светло улыбаясь, возразил арестант, <b>[yesua5]</b></p>

– <b>[hegemon5]</b> – улыбнувшись, сказал Пилат, – <b>[hegemon6]</b> После этих слов у него, уже в светлой голове, отчетливо сложилась формула приговора. И он ее тут же озвучил для записи в протокол: игемон разобрал дело бродячего философа Иешуа и состава преступления в нем не нашел.</p>

-----

hegemon1 = "Так ты утверждаешь, что не призывал разрушить... или поджечь, или каким-либо иным способом уничтожить храм?";

hegemon2 = "Так поклянись своей жизнью, что этого не было,";

hegemon3 = "Ею клясться самое время, так как она висит на волоске, помни это.";

hegemon4 = "Но я легко могу перерезать этот волосок.";

hegemon5 = "Так, так";

hegemon6 = "теперь я не сомневаюсь, что праздные зеваки в Ершалаиме ходили за тобой по пятам.";

-----

yesua1 = "Я, игемон, никогда не призывал к подобным действиям, повторяю.";

yesua2 = "Не думаешь ли ты, что ты ее подвесил, игемон?";

yesua3 = "Если это так, то ты сильно ошибаешься.";

yesua4 = "И в этом ты ошибаешься,";

yesua5 = " – согласишься, что перерезать волосок уж наверно может лишь тот, кто подвесил ?";